An Exploratory Study of Open Source Software Based on Public Project Archives

Neng Xu

A Thesis
in
The John Molson School of Business

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Science in Administration at
Concordia University
Montreal, Quebec, Canada

April 2003

ABSTRACT

An Exploratory Study of Open Source Software Based on Public Project Archives

Neng Xu

This thesis conducts an exploratory study of Open Source Software (OSS) from various perspectives in order to discover and demonstrate fertile research areas in OSS that can benefit from the public archives of OSS projects. It follows a horizontal research method which combines theoretical model building with empirical data analysis.

On the theoretical side, it classifies existing quantitative OSS studies and categorizes the public archives. It defines the concept of an OSS project by delineating its four critical components – community, methodology, products, and services. It specifies the roles in OSS communities, examines the speed, cost, and quality of OSS development, and reveals the impacts of programming languages on software projects. Most importantly, it originates a new approach to OSS adoption research which comprises strategic level study of OSS adoption and an assessment framework for OSS projects. A rich set of propositions are formulated for future study.

On the empirical side, it analyzes summary statistics of 48,331 OSS projects and more detailed attributes of 1,907 projects which use Python as one of their programming languages. It depicts the portraits of OSS projects in general, and Python projects in specific. Caveats, for pitfalls and weaknesses of OSS projects discovered from the analysis, such as small development teams and competing projects, are announced with suggestions for improvements.

Inspiring original ideas into a burgeoning research domain, this thesis contributes to both comprehension and practice of OSS.

This thesis is dedicated to my wife and my parents, who are the wind beneath my wings.

ACKNOWLEDGEMENTS

I am enormously grateful to Dr. Jamshid Etezadi, my supervisor, for his continuous guidance and help throughout the years.

I would like to express my appreciation to Dr. El-Sayed Abou-Zeid, and Dr. Rustam Vahidov, members of my thesis supervisory committee, who helped me a lot to refine this thesis.

My great gratitude goes to all OSS / Free Software researchers, especially to Rogerio Atem Carvalho, Sandeep Krishnamurthy, Stefan Koch, Greg Madey, and Michael Godfrey, who generously offered many advices to me for this thesis via emails, though we did not meet or talk even once.

I would like to thank the support team of SourceForge.net, who kindly provided data for this thesis.

Last but certainly not the least, my hearty respect goes to all OSS / Free Software developers for their altruistic contributions that are changing the world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Open Source Software (OSS) is a strong competitor in many software application areas. OSS products have become inconvincible challengers to proprietary closed source software products (Raymond 2001). The essence of OSS is the free sharing of source codes. Anyone can obtain, modify, and distribute (with or without modifications) the source codes of OSS products. The availability of source codes is guaranteed for an OSS product. Most OSS projects use the popular Concurrent Versions System (CVS) to keep the history of code evolution (van der Hoek, 2000).

Furthermore, since most OSS projects involve numerous developers, testers, technical writers, and users who are scattered all over the world, virtually all communications among people are taking place in cyberspace via communication tools such as email, mailing lists, news groups, forums, and the web. Thus almost all information on an OSS project are recorded in electronic form and archived to keep the whole history (Mockus et al., 2000; Robbins 2002). These electronic tools and archives can be freely accessed in real-time by anyone interested to obtain up-to-date information on an OSS project (Robbins 2002).

In stark contrast to the paucity and difficulty of obtaining data for proprietary software, these publicly accessible archives then become a free treasure trove for researchers (Chandra & Chen, 2000). "Free" has two connotations here, like traditionally quoted in OSS literature (e.g., Dempsey et al., 2002; DiBona et al., 1999), one as in "free speech", and another as in "free beer". Any researcher can access the same information without charge. So results from one study can be replicated and confirmed by other

researchers. Moreover, diverse analyses can be carried out from assorted perspectives by different researchers. Existing studies on Linux kernel (e.g., Antoniol et al., 2002; Godfrey & Tu, 2000; Schach et al., 2002) demonstrate the convenience of carrying out research based on public archives of OSS project.

Conversely, source codes, communication exchanges, bug reports, and other project information of proprietary software are often kept as secret from public access. In particular, most companies are reluctant to publish information on faults or bugs of their software (Chandra & Chen, 2000; Pfaff 1998). Though some researchers may get special permission to access proprietary information, it would be a pain for other researchers to get the same access in order to justify the research, let alone to carry out analysis from other perspectives.

There are already pioneering researchers who begun their exploration into the public archives of OSS projects (e.g., Antoniol et al., 2002; Capiluppi 2002; Chandra & Chen, 2000; Godfrey & Tu, 2000; Crowston & Scozzi, 2002; Gonzalez-Barahona et al. 2002; Koch & Schneider, 2002; Krishnamurthy 2002; Lakhani & von Hippel, 2002; Madey et al., 2002a; Madey et al., 2002b; Mockus et al. 2000; Schach et al., 2002; Stamelos et al., 2002; Wheeler 2002a). The approach of exploring historical information from OSS public archives solved one of the most serious problems in software engineering research – the lack of data, especially historical data (Koch & Schneider, 2002). However, as revealed by the publishing dates of these pioneering studies, this kind of exploration is still at its nascence.

This thesis conducts an exploratory study of OSS from various perspectives based on the public archives of OSS projects. Its primary research objective is to discover and

demonstrate fertile research areas in OSS that can benefit from the public archives of OSS projects, with special emphasis on OSS adoption study.

Suitable for its primary research objective and its exploratory trait, this thesis is in favor of a horizontal research method which combines theoretical model building with empirical data analysis. It discusses many related topics in various depths without the intention of being exhaustive or exquisite on any specific topic. Abiding by this horizontal research method, this thesis is conducted in the following logical steps:

1. Review existing quantitative research based on the public archives of OSS projects, and propose a classification mechanism for quantitative OSS studies. Literature review is always the first step to take for a research. Although qualitative research can also be carried out on the public archives, it is quantitative research that can benefit the most from the archives. Since there is only limited existing quantitative research as shown in Section 2, exploratory research that discovers and demonstrates fertile research areas is much required.

2. Categorize the public archives of OSS projects and provide examples of research based on the various types of archives. Researchers should know the types and contents of the public archives before they can use it. And examples of existing research which utilized the public archives can bring insights to future research.

3. Study the entity which generates the public archives – OSS projects. This thesis suggests an initial definition of an OSS project and delineates the components of an OSS project. Horizontal discussions on various topics are related to the components of an OSS project.

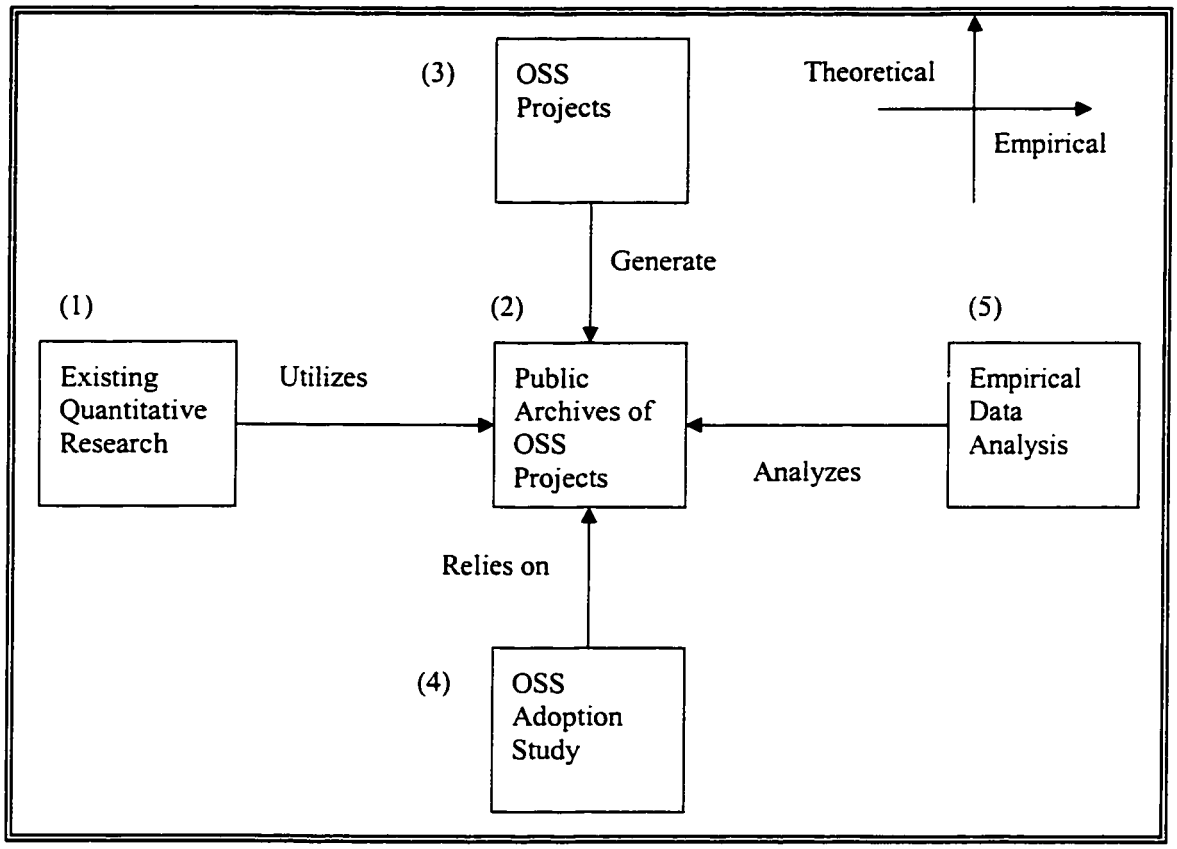
4. Study the usage of OSS and the people who use the OSS – OSS Adoption and OSS adopters. This thesis pinpoints the deficiency of some existing IT adoption models, and proposes a novel approach to OSS adoption research which relies on the public archives. Although theoretical model building is woven into every preceding step (e.g., classification schemes and definitions), this step is a more in-depth manifestation of the theoretical model building part of the horizontal research method of this thesis.

5. Conduct an empirical data analysis to depict the characteristics of a rich set of OSS projects, which have not been studied in previous studies. This step furnishes, of course, the empirical data analysis part of the horizontal research method of this thesis.

Figure 1 depicts the major entities studied in this thesis and their relationships at highly abstracted level. The numbers in parenthesis indicate the logical steps of this thesis as discussed above.

As the core study entity of this thesis, the public archives of OSS projects are at the origin of the two coordinate axes. The X axis is the empirical axis which determines the breadth and extensiveness of this thesis. Along this empirical axis, there are existing quantitative research which utilizes the public archives and a new empirical data analysis which analyzes the public archives. The Y axis is the theoretical axis which determines the altitude and depth of this thesis. Along this theoretical axis, there are OSS projects which generate the public archives and the OSS adoption study which relies on the public archives. The attributes of these various entities and their relationships will be delineated at more fine-grained levels in later sections.

Figure 1: The structure of this thesis



The organization of this thesis is as follows. Section 2 summarizes existing quantitative studies on OSS and suggests a classification mechanism of these studies. Section 3 provides a categorization of the public archives, describes the major characteristics of these archives, and presents examples of previous studies which used these archives as their data sources.

Section 4 suggests an initial definition of an OSS project by delineating the four critical components of an OSS project – community, methodology, products, and services. Major discussions in this section cover the specification of roles played by OSS community members, OSS development model and software crisis, impacts of

programming languages, documentary structure of source code, and business models based on OSS.

Section 5 and Section 6 conduct a study of OSS adoption from new perspectives. This study comprises two levels: strategic level and tactical level. Section 5 carries out the strategic level study of OSS adoption, and Section 6 carries out the tactical level study of OSS adoption. Major contributions include the contents of OSS adoption, two novel perspectives for adoption study, the Strategic Usages of OSS, an innovative classification of OSS adopters, and a comprehensive evaluation framework of OSS projects based on the public archives. Relevant propositions are formulated in Section 6 for future study. Empirical research questions are also posed in Section 6, to be answered in Section 7.

Section 7 carries out an empirical data analysis on a rich set of data collected from OSS projects. A wide variety of data describing characteristics of OSS projects as of 3 October 2002 are collected from public archives hosted at SourceForge.net (http: //sourceforge.net), which is the largest repository of OSS projects in the world. These data include summary statistics of all 48,331 projects hosted at SourceForge.net, and more detailed attributes of 1,907 projects which use Python as one of their programming languages. Various analyses of these data depict the portraits of OSS projects in general, and Python projects in specific. The results of data analysis are interpreted in the context of the assessment framework suggested in Section 6. Some caveats for pitfalls and weaknesses of OSS projects discovered from the empirical data analysis are also announced with suggestions for improvements. Finally, Section 8 summarizes the contributions, implications, and limitations of this thesis.

2 Quantitative Research on OSS

An extensive literature search for recent studies on OSS and relevant topics was conducted via various sources including the ProQuest database, the Digital Library of the Association for Computing Machinery (ACM), and the Internet. Since OSS is still a new research area, not much research has been published in academic journals (Feller & Fitzgerald, 2001). Especially, there are only a limited number of existing quantitative studies on OSS projects (Crowston & Scozzi, 2002). Just like the subject – OSS – itself, Internet is the major publishing resource for research on OSS. DiBona et al. (1999) described this phenomenon precisely: The Internet is the printing press of the digital age. There are two web resources that offered great help: the Free / Open Source Research Community (http: //opensource.mit.edu) and the Open Source Resources (http: //opensource.ucc.ie).

2.1 Limited Existing Research

Traditional data collection methods such as surveys are still utilized in studies on OSS. However, many surveys are now conducted in electronic means such as web survey or email survey, rather than using the traditional mail. The web-based study on motivation in Linux community is a famous example (Hertel et al., 2002). Zhao and Elbaum (2000) conducted a small-scale survey on quality related activities in open source projects via email. Jorgensen (2001) used a web-based questionnaire and followed it up with interviews by email to collect data from developers of the FreeBSD project.

This trend of adopting modern communication tools in research is in line with the spirit of OSS, since most communications within an OSS project are carried out via

Internet. For a well maintained OSS project, almost all its historical information is conserved in various forms of archives that are freely accessible by anyone interested. These archives then become a valuable resource for OSS researchers.

Unfortunately, very few quantitative studies based on the public archives of OSS are published in academic journals. This kind of studies found in the literature review include Antoniol et al., 2002; Crowston and Scozzi, 2002; Dempsey et al., 2002; Koch and Schneider, 2002; Schach et al., 2002; and Stamelos et al., 2002. Notably, four out of six of them are published in special issues on OSS of two journals: special issue on Open Source Software Engineering of IEE Proceedings - Software (Vol. 149, No. 1., February 2002), and a double special issue on Open Source Software of the Information Systems Journal (Vol. 11, No. 4 / Vol. 12, No. 1, October 2001 / February 2002). Hence, quantitative research on OSS just starts to catch the attention of academic journals. This situation is an interesting contrast to another topic – the Technology Acceptance Model (TAM) (which is discussed in later sections). Plenty of quantitative studies based on the TAM are published frequently in academic journals. For example, in one of the top journals – the MIS Quartly, this kind of studies debuted with the seminal article of Davis (1989) and followed by quite a few articles like Szajna (1994) and Karahanna et al. (1999), across a long time span.

Several quantitative studies based on the public archives of OSS are presented at international conferences (e.g., Chandra & Chen, 2000; Godfrey & Tu, 2000; Madey et al., 2002b; Mockus et al. 2000). And some of the presentations are in the form of position papers (e.g., Capiluppi et al., 2002; Godfrey & Tu, 2001; Madey et al., 2002a). There are

also some working papers posted in the web (e.g., Gonzalez-Barahona et al. 2002; Krishnamurthy 2002; Lakhani & von Hippel, 2002; Wheeler 2002a).

About half of the above mentioned quantitative studies are individual case studies based on one to three OSS projects. They covered only a few high-profile OSS projects such as Linux and Apache. Due to the limitation of number of projects investigated in these studies, their results cannot be generalized to the population at large. Furthermore, Crowston and Scozzi (2002, p. 7) pointed out: "attention only to the largest and most successful OSS projects provides a misleading view of the general state of OSS development".

The other half of these studies collected data from a larger number of OSS projects (usually greater than 99), but focused on only a few features of OSS projects such as the number of developers or size of source codes. Cross-sectional analyses (mostly using descriptive statistics and correlational analysis) were the norm among these studies.

As revealed in the above concise literature review, the assets of the public archives of OSS projects are far away from being fully discovered. There is a need to empirically explore further and to analyze more comprehensively the public archives of a rich set of OSS projects, which have not been studied in previous studies. Moreover, there is also a need to build up more theoretical models that take advantage of the wealth of the public archives. Accordingly, the public archives of OSS projects become the core entity studied in this thesis. And all other major entities studied in this thesis as stated in Section 1 are closely related to the utilization of this core entity.

2.2 Classification Mechanism

Due to the limited amount of existing quantitative academic studies on OSS, there is no meta-analysis done to classify these studies. But classification mechanisms for research have been proposed in relevant fields. A recent example of these mechanisms is Glass et al. (2002), in which a comprehensive classification scheme was presented to examine research in Software Engineering. This scheme sorts relevant research papers by five characteristics: the topic of each paper, its research approach, its research method, the reference disciplines it utilized, and the level at which it addressed the issues. There are about 50 topics covering the entire computing field. The research approach can be descriptive, evaluative, or formulative, with sub-categories for each major approach. Twenty-two research methods are identified, ranging from conceptual analysis to simulation. There are twelve reference disciplines including some possible overlaps such as management science / management / public administration. And the ten levels / units of analysis (e.g., profession, abstract concept, etc) are not positioned in a finely structured hierarchy.

As comprehensive as it seems to be, this scheme is practically too complex for categorizing the small group of quantitative research on OSS. The result of using this scheme on existing quantitative research on OSS may resemble the effect of a scarce matrix: there are simply not enough studies so most of the cells in the matrix are just null. A simple but effective classification mechanism could be more appropriate. This thesis is an initial attempt to propose such a classification mechanism which uses one simple characteristic – data source – as the classification criterion. Because data is the basis of

quantitative research, this simple classification criterion can highlight the research design of previous studies and the generalizability of their results.

According to the number of OSS projects from which the studies collect data, quantitative OSS studies can be classified into four categories: Focus Project Study, Comparative Project Study, Sample Project Study, and Population Project Study. In a Focus Project Study, data are collected from only one specific OSS project. Several projects become the data source of a Comparative Project Study. A Sample Project Study collects data from a sample selected from a specific population of OSS projects. The whole entirety of a specific population of OSS projects becomes the date source for a Population Project Study.

This simple classification scheme is not meant to be exquisite but practical. In fact, it can effectively and conveniently classify existent quantitative research on OSS. When the OSS research society accumulates a much richer set of quantitative studies in the future, a more complex and comprehensive classification scheme like that in Glass et al. (2002) could be deployed to handle the much larger quantity and variety of quantitative studies on OSS.

2.2.1 Focus Project Study

Focus Project Study concentrates on one specific project as the target of study. This kind of narrow focus obviously brings its advantage as well as disadvantage. On the advantage side, since only one project is fathomed, more fine-grained data collection and data analysis can produce more comprehensive and detailed understanding of the project. On the disadvantage side, the results of a Focus Project Study can hardly be generalized.

As the flagship of OSS projects, Linux is possibly the most frequently studied project. Godfrey and Tu (2000) investigated the growth rate of Linux kernel. Antoniol et al. (2002) inspected the extent and evolution of code duplication in Linux kernel. Similarly, Schach et al. (2002) examined the code coupling in Linux kernel.

Koch and Schneider (2002) chose the GNOME project as their target. GNOME is a desktop environment for Linux. They made a total effort estimation of the project by examining the effort already expended by the participants of the project in the forms of source code contribution as well as posted messages in the mailing lists.

Other OSS projects examined in previous focus case studies are also high profile projects. For example, Mockus et al. (2000) studied developer participation, core team size, productivity, and defect density in the Apache project.

2.2.2 Comparative Project Study

Comparative Project Study typically selects threesome projects as the target of study. It is a compromise between Focus Project Study and Sample Project Study. On the advantage side, it may highlight useful nuances by comparing and contrasting the selected projects. Thus it may gain more insights than Focus Project Study. On the disadvantage side, the workload of research can be multiplied compared to Focus Project Study. And the generalizability of findings is still stringently restricted.

Chandra and Chen (2000) fathomed types of programming faults that occurred in three OSS projects: Apache web server, GNOME desktop environment, and MySQL database. Payne (2002) assessed the security of three Unix-based operating systems: Sun Solaris, Debian GNU/Linux, and OpenBSD, with the latter two being open source.

2.2.3 Sample Project Study

Sample Project Study selects a large number of projects (usually > 99) using various sampling methods from a definite population of projects as the target of study. On the advantage side, statistical analyses can be applied across the sample. Moreover, its results can be generalized to the population with some extent of validity. On the disadvantage side, due to time, budget, and effort constraints, coarse-grained data collection may be more suitable. Thus, it may miss the variances and insights visible only at more fine-grained levels as in Focus Project Study and Comparative Project Study. There are a few examples in Section 7 where further fine-grained analysis into individual projects are required to explain the phenomena shown by the coarse-grained statistics collected over a large number of projects.

Stamelos et al. (2002) randomly selected 100 C applications from the SuSE Linux 6.0 release. They used a software measurement tool called Logiscope to analyze these applications, calculate values of certain metrics, and provide recommendations for code improvement. These 100 applications were assessed against a programming standard provided by Logiscope itself. The authors deemed this standard as the result of empirical conclusions that came out after the analysis of millions of lines of industrial source code. Thus, the quality of these applications was defined as the conformance to the acceptable range of values, as set in Logiscope, for certain metrics.

Wheeler (2002a) and Gonzalez-Barahona et al. (2002) investigated the size of two GNU / Linux distributions: Red Hat 7.1 and Debian 2.2, respectively. Their samples are restricted by the tool "sloccount", developed by Wheeler for counting lines of code. Only

those programs which are written in one of the programming languages supported by sloccount are processed and analyzed.

Possibly due to the heavy workload and inherent complexity, there is no Sample Project Study which carried out time series analysis, e.g., to study the evolution of releases over time, across the sample of OSS projects. This is a niche that encourages new research.

2.2.4 Population Project Study

Population Project Study is a special case of Sample Project Study where the sample is the whole population. In another word, there is no sampling for the population. After providing a clear definition of a population, a Population Project Study collects data from all elements of this population. Of course, this population should be definite and accessible. The vital advantage of a Population Project Study is its generalizability since the whole population is studied directly. Nonetheless, it faces also the same kinds of disadvantages like in a Sample Project Study.

It should be noted that the population in a Population Project Study could be much narrower in scope than a population in a Sample Project Study. For instance, the former could be all OSS projects hosted at one web site while the latter could be all OSS projects in the world. In this case, all OSS projects hosted at one web site are the population for a Population Project Study. But meanwhile, they can be treated as a sample taken from the bigger population – all OSS projects in the world. Hence, a Population Project Study can be considered as a Sample Project Study in the sense that the population in a Population Project Study is a sample from a bigger population.

Crowston and Scozzi (2002) did a Population Project Study in which they collected data from all 11959 OSS projects hosted at SourceForge.net. In another word, their population is all projects hosted at SourceForge.net. Their final size of population was fairly large: 7477 projects with no missing values. They sought to find critical success factors of OSS projects using competency rallying theory, of which Crowston is one of the authors. Albeit the best adjusted R^2 from their regression analysis was only .476, they still deemed that their results of data analysis were satisfactory. Understandably, their conclusions were not convincing enough.

2.3 Summary

There is only limited quantitative research which utilizes the public archives of OSS projects. Section 2 proposed a simple but effective classification mechanism for quantitative OSS studies which uses data source as the classification criterion. Accordingly, quantitative research on OSS can be categorized into Focus Project Study, Comparative Project Study, Sample Project Study, and Population Project Study. Table 1 shows this classification mechanism.

Table 1: A classification mechanism for quantitative OSS studies

Category	Data Source
Focus Project Study	One project.
Comparative Project Study	Several projects.
Sample Project Study	A sample of projects from a population of projects.
Population Project Study	One whole population of projects.

3 Public Archives of OSS Projects

Because the public archives of OSS projects are a valuable resource for researchers, it is reasonably imperative to know what kinds of archives exist and what kinds of contents they maintain. Above mentioned quantitative studies on OSS did not provide a classification of these archives. They assumed that these archives were known to general audience. And non-uniform names were used for these archives. For example, Koch and Schneider (2002) called mailing lists as "discussion lists". Moreover, they only used one or two kinds of archives such as mailing lists or CVS. So it is hard to appreciate the richness and comprehensiveness of these archives from previous studies.

A classification of the public archives together with descriptions of their characteristics and examples of previous studies are helpful to future researchers who are not familiar with these archives. This thesis proposes a classification of public archives for OSS. The characteristics of each archive are briefly introduced, together with examples of existent research that utilized the archive. The major archives are: Software Releases, Configuration Management Systems, Bug / Issue Tracking Systems, Mailing Lists, News Groups, Forums, Documentation, and Others.

3.1 Software Releases

Unlike proprietary software, the releases of an OSS project must provide all source codes. Often, they also include binary packages for popular platforms, as well as pertinent documentation. Although not all OSS projects keep an archive of all releases, the most recent release should always be available on line.

A phenomenal kind of software release is the competing GNU / Linux distributions by various distributors including Red Hat, Debian, Slackware, Mandrake, SuSE, Caldera, etc. As of 28 September 2002, there are 166 different distributions listed at the official Linux site (www.linux.org). Each distribution comprises several hundreds of applications on top of the Linux kernel.

Godfrey and Tu (2000) is a typical example of Focus Project Study based on public archive of software releases. The authors examined 96 Linux kernel releases in order to depict the growth of Linux kernel. It is a kind of time series analysis since it covers a certain period in the life span of a project. The author of this thesis asked Mr. Godfrey about the reason why they chose only 96 kernel versions instead of studying all available versions. Mr. Godfrey replied that it would not be likely to find any more useful insights by measuring all several hundred releases.

Similarly, Antoniol et al. (2002) inspected 19 releases of Linux kernel for identifying code duplication. Their sample size is even less than that of Godfrey and Tu (2000). It is unknown if they chose this small sample size due to the same reason as provided above by Mr. Godfrey. Contrariwise, Schach et al. (2002) adopted a much larger sample size by examining 365 versions of Linux kernel.

Stamelos et al. (2002) is a typical example of Sample Project Study based on one specific release of each project across a sample of projects. It is a kind of cross-section analysis since it only covers one specific point of the time span of a project. Similarly, Wheeler (2002a) and Gonzalez-Barahona et al. (2002) investigated the size of two GNU / Linux distributions.

3.2 Configuration Management Systems

Configuration management tools play a central role in OSS projects. Without the support of configuration management tools, it is impossible to coordinate the cooperation of distributed developers in producing and updating coherent codes for a project. There are already some studies on the practice of configuration management in OSS projects. For instance, Asklund and Bendix (2002) suggested that lessons learned from OSS configuration management practices could be applied to closed source software projects.

As the general repository of source codes and documentation, the Configuration System is a paramount archive that deserves much attention. Many sorts of information can be extracted from this kind of archive. For example, the sizes of software modules over time can be used to depict the evolution of the project over time.

CVS is the prevailing configuration system and version control system used by most OSS projects, especially those complex ones, despite the fact that it is losing comparative advantages to commercial configuration systems (Robbins 2002; van der Hoek, 2000). There are many third party add-ons for CVS: ViewCVS, Bonsai, CVSWeb, etc. These add-ons provide Web interface to the console-only CVS.

Every commit to CVS can automatically generate an email message sent to a dedicated mailing list such as the "Source Change Reports" mailing list in the Apache project. Thus accessing the archive of that mailing list is an alternative to retrieving information directly from CVS. Mockus et al. (2000) used this method to reconstruct the CVS data.

Koch & Schneider (2002) is a typical example of Focus Project Study that retrieved data directly from the CVS repository. For every check-in of the GNOME CVS, the authors extracted data for programmer, file, date, lines of code added and deleted, revision number, and comments.

3.3 Bug / Issue Tracking Systems

Trackers are tools to help the OSS community to keep record of bugs, patches, support requests, feature requests, and other issues. Named "Linus's Law", "Given enough eyeballs all bugs are shallow" is one distinguishing characteristic of OSS promoted by Raymond (2001). Peer-review and parallel debugging are highly emphasized in OSS development model (Feller & Fitzgerald, 2000). Bug reports by community members including developers as well as users are thus quite important to the success of an OSS project. Popular bug tracking software used in OSS projects includes Bugzilla, GNATS, DebBugs, and RequestTracker (Halloran & Scherlis, 2002).

Bug reports normally contain the symptoms of the faults, the results of the fault, and the operating environment and workload that induce the fault (Chandra & Chen, 2000). Some bug reporting systems, such as GNATS and Bugzilla used in Apache, are set up to generate an email message to a dedicated mailing list for each transaction and saved in a separate mailing list archive. The message includes various information such as problem report number, affected module, status, name of the submitter, date, and comments (Mockus et al., 2000).

Chandra and Chen (2000) deemed that in most cases bug reports also include how the underlying bug was fixed. However, it is not true according to my observation over several projects at SourceForge.net. On one hand, there are cases where the patches to fix

the bug were attached to the bug report. On the other hand, in many cases, the indication of fix is simply like "This bug has been fixed in CVS and will appear in the next release". It is not helpful at all for knowing what the exact changes are.

Linking details of bug fixes to bug reports can greatly facilitate the analysis of bug tracking and fixing. For example, if defect fixes often require changes to large numbers of source files, then the architecture of the application is probably not designed well. And it is then possible to determine how many defects have resulted in modifications to a particular module (Godfrey & Tu, 2000). From another perspective, if a module is often changed by defect fixes, it is probably not designed well. Project Management team can then focus their effort on improving this kind of hot spot of defects. Unfortunately, not all project or all developers abide by the best practice to furnish the linking.

Chandra and Chen (2000) were lucky in only one of those three projects they studied. Apache seemed to provide enough information in their bug reports. So they can manually analyze the bug reports and classify the types of faults as well as the types of fixes. For GNOME, they had to also check the CVS to find out how the bugs were fixed. For MySQL, the archive of mailing list was used as the source of information instead.

Many other projects may not provide the facility of linking details of fixes to bug reports either. For example, even the flagship Linux did not provide such detailed change logs (Godfrey & Tu, 2000). It would be then difficult to make full use of bug reports in these projects.

Other tracking systems for patches, support request, feature request, and other issues were not studied in previous research.

3.4 Mailing Lists

Mailing lists are convenient tools for asynchronous communication. The volume of the Apache mailing list is about 40 messages a day (Sandred 2001, p. 147). Mailing list is the traditional communication tool preferred by OSS developers. However, it is not efficient by its "push" design: it unnecessarily increases network circulation by sending every email to all subscribers, even when most subscribers may discard many of the emails. It could suffer from spam if not controlled tightly. The author of this thesis found an example in the ViewCVS developer mailing list (http: //mailman.lyra.org /pipermail /viewcvs-dev/), where spam caused even the core developer to unsubscribe. Nowadays, most people's email boxes are always bombarded by junk mails. It is then a little surprise that mailing lists are still a favorite choice of many OSS developers.

Another deficit of many mailing lists is the lack of search functionality. Since the contents of mailing lists are sent to a subscriber in the form of emails, the subscriber has to use external software or to manually scan the series of emails in order to search for some contents. Although mailing lists are usually archived by software like pipermail, not all archives provide search functions like keyword search or other advanced search options which are a norm in other kind of communication tools like web-based forums. To look for specific information in those without-search-support archives of mailing lists is not an easy task.

Popular mailing list software used in OSS projects includes Mailman, Ezmlm, MHonArc, and Majordomo (Halloran & Scherlis, 2002). There are many different kinds of mailing lists for various purposes: developers, users, news, etc. As discussed in Section 3.2 and Section 3.3, there are also dedicated mailing lists for changes in source

and bug reports, whose messages are automatically generated by configuration management system or bug tracking system.

Qualitative content analysis may be used to analyze mailing lists. On the quantitative analysis side, Mockus et al. (2000) is a typical Focus Project Study that used mailing lists as the sole data source. The authors studied three mailing lists of the Apache project. They extracted date, sender identity, message subject, and details on code changes and problem reports from the Development mailing list. From the Source Change Reports mailing list, they extracted date and time of the change, developer login, files touched, numbers of lines added and deleted for each file, people who submitted and / or reviewed the change, and the problem report number for changes made as a result of a problem report. From the Bug Database Reports mailing list, information such as problem report number, affected module, status, name of the submitter, date, and comments were extracted.

Koch & Schneider (2002) is a typical example of Focus Project Study that used mailing list as one of data resources. The authors extracted sender, subject, time, and complete text from relevant mailing lists of the GNOME project.

3.5 Newsgroups

Newsgroups used to be another preferred method of communication by traditional OSS developers. It uses a "pull" design: reader chooses which news to download and read. Thus, it is more efficient than mailing lists. Having topics as rich as the mailing lists, it uses a different protocol NNTP (Network News Transfer Protocol) other than HTTP as in forums. However, newsgroups seem to be loosing ground to the web-based forum. One interesting fact is that in October 2002, SourceForge.net started to use NNTP

as an alternative of HTTP to view and post messages to forums. And the support of mailing lists using NNTP is also planned. So it is possible that newsgroups will be merged into forums. But the NNTP protocol may find new application areas.

Qualitative content analysis may be used to analyze newsgroups. Though unpublished, Lakhani and von Hippel (2002) is the most cited quantitative OSS research on the use of newsgroups. The authors analyzed the characteristics of user-to-user assistance occurred in a newsgroup for the Apache project.

3.6 Forums

Web-based forums are a novel means of communication. They are quite popular outside the developer circle, i.e., in ordinary Web surfers. Unlike the mailing lists, forums adopt the "pull" mechanism to impart information and usually incorporate advanced search functions. Unlike the newsgroups, forums speak the more popular protocol – HTTP. Because of their comparative advantages over mailing list and newsgroups, forums could become the next prevailing communication tool for OSS community.

Famous forms of forum include SlashDot (http://slashdot.org), Wiki (http://wiki.org), Ultimate Bulletin Board (http://www.infopop.com), etc. Among them, Wiki is a very interesting form which acts like an electronic whiteboard.

Qualitative content analysis may be used to analyze forums. There is not yet well-known quantitative research on OSS forums. However, research has been done on other kinds of forums. For example, Zhang and Storck (2001) studied the important role peripheral members played in a travel forum hosted by a major Internet information service company in China.

3.7 Documentation

Documentation used to be the weakest link of OSS projects. There is no standardized archive for documentation in OSS projects. And the quality and quantity of documents vary across projects. However, many projects maintain dedicated areas for documentation on their web sites. And SourceForge.net also provides such dedicated area, but not all projects use it. Frequently Asked Questions (FAQ) is a popular form of documentation. FAQs are generally supplied in OSS projects.

There is no existent quantitative research on documentation of OSS projects in the reviewed literature.

3.8 Others

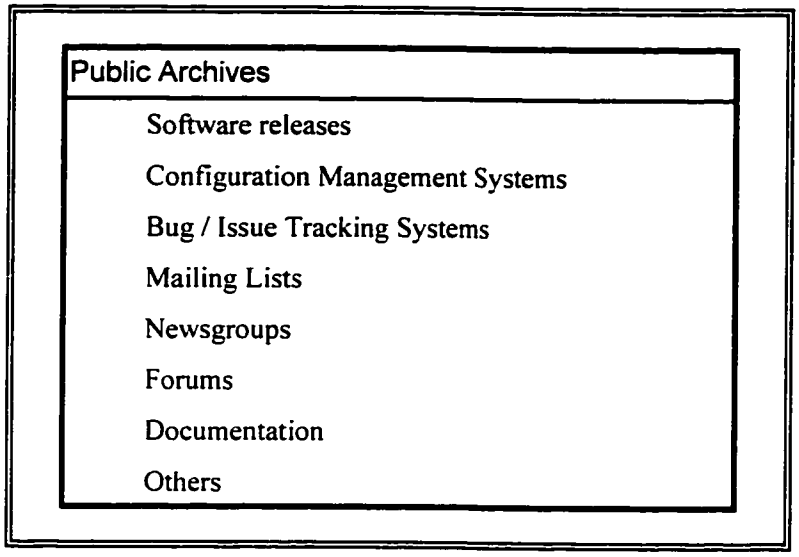
There are other kinds of archives used in OSS projects. News or announcements are for the disclosure of important new events. Surveys can be conducted on the project Web site. Sometimes, subprojects may be set up to deal with specific tasks.

There is no existent quantitative research on other archives of OSS projects in the reviewed literature.

3.9 Summary

Section 3 presented a categorization of public archives of OSS projects, with indication of potential usage of these archives by exampling pertinent previous studies. Figure 2 shows this categorization.

Figure 2: A categorization of public archives of OSS projects



4 OSS Projects

Though the concept of project is well known as defined by PMI (2000), no formal definition of an OSS project is provided in reviewed literature. This thesis makes an initial definition.

OSS project: A kind of project where an OSS community follows a methodology that adheres to the OSS development model, to produce software products that are conformant to the definition of OSS, and to provide relevant services.

This initial definition reveals the critical features of an OSS project. The vital feature of an OSS project is, of course, that the software products produced in this project should be conformant to the definition of OSS. The Open Source Initiative (http: //www.opensource.org) is the authority who made the Open Source Definition (Appendix A). A software can be deemed as Open Source Software only when it is released under a license that conforms to the Open Source Definition (Feller & Fitzgerald, 2000).

Another paramount feature of an OSS project is that the methodology deployed in the project should adhere to the OSS development model. The OSS development model is characterized by voluntary participation, peer-to-peer collaboration, Internet based communication, and decentralized management. Moreover, the history of an OSS project is recorded in the public archives of the project.

Other critical features of an OSS project are the community and the services. Hence, an OSS project has four critical components: community, methodology, products, and services. The following sections will highlight the attributes of these four components, and discuss many valuable topics associated with these components in detail.

4.1 OSS Community

The community is the people and entities (companies, organizations, etc.) who participate in the OSS project. Obviously, the community is the source of power and wisdom of an OSS project. An OSS community comprises all stakeholders in an OSS projects: developers, users, service providers (such as consultants, book publisher, etc.). It is more than the project team in the normal commercial project environment. Though stakeholders of a project are taken into consideration by the theory of project management (PMI 2000), it is in an OSS project that all stakeholders become an integral part of the project.

As Raymond (2001) encapsulated in the famous metaphor "scratching an itch", most OSS projects started out with one developer who wanted to solve one particular problem that could not be solved satisfactorily by existing software according to this developer's judgment. More importantly, this original developer adopted the idea of OSS by making her source codes publicly available in the Internet. So interested people can easily try out the software by downloading and compiling source codes, and contribute back by reporting bugs, submitting patches, and adding new functions. Collaborative development within a community connected via Internet is one of the essential characteristics of OSS (Nakakoji et al., 2002). Without this kind of community, OSS projects are not likely to be successful (Nakakoji et al., 2002).

This thesis does not discuss in detail why people contribute to an OSS project because there are many previous studies on this topic. For a narrative style, Raymond (2001) is an influential figure in the OSS movement. For an empirical style, Hertel et al. (2002) carried out online survey of Linux kernel contributors. Although all previous

studies provide some insightful notions, this thesis suggests that the most important idea is to recognize that all members of OSS community act on their own free will. There could be as many reasons as the number of the community members since any member could have his / her own special motivation. Thus, the motivation study of OSS community should be conducted at individual level rather than group level.

The members of OSS community voluntarily join the community and contribute to the OSS project in various ways. So it is natural to define the roles played by the members as per their contributions. The following sections will criticize the existing classification of roles and propose a refined new classification.

4.1.1 Existing Classification of Roles in Community

Previous studies suggested various classifications of roles. Some used coarse-grained classes. For example, suffice to their purpose, Koch and Schneider (2002) distinguished only two kinds of people participated in the GNOME project. One type is coders who submitted check-in files to CVS repository. The other type is posters who posted messages to mailing lists. However, they indicated that the top 52 coders supplied 80% of commits, while 11 coders were even more active among those top 52 coders. The existence of these top contributors suggests that fine-grained classes could be more useful. Markus et al. (2000) also found that a few core developers donated the majority of codes in the Apache project.

In the reviewed literature, the classification framework proposed by Nakakoji et al. (2002, p. 79) is the most sophisticated definition of roles in OSS community. They defined eight roles:

Passive User. Passive Users just use the system in the same way as most of us use commercial software; they are attracted to OSS mainly due to its high quality and the potential of being changed when needed.

Reader. Readers are active users of the system; they not only use the system, but also try to understand how the system works by reading the source code. Readers are like peer reviewers in traditional software development organizations.

Bug Reporter. Bug Reporters discover and report bugs; they do not fix the bugs themselves, and they may not read source code either. They assume the same role as testers of the traditional software development model.

Bug Fixer. Bug Fixers fix the bug that is either discovered by themselves or reported by Bug Reporters. Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs.

Peripheral Developer. Peripheral Developers contribute occasionally new functionality or features to the existing system. Their contribution is irregular, and the period of involvement is short and sporadic.

Active Developer. Active Developers regularly contribute new features and fix bugs; they are one of the major development forces of OSS systems.

Core Member. Core Members are responsible for guiding and coordinating the development of an OSS project. Core Members are those people who have been involved with the project for a relative long time and have made significant

contributions to the development and evolution of the system. In some OSS communities, they are also called Maintainers.

Project Leader. Project Leader is often the person who has initiated the project. He or she is responsible for the vision and overall direction of the project.

Nakakoji et al. (2002) stated that the influences the members have in the community depended on what role they played. The above role classification implies the influences of these roles in ascending order, with the Passive User having the least influence and the Project Leader having the most influence. It also implies the quantity of members in each role in descending order, with most members being Passive Users and the percentage dropped sharply from Readers to Core Members.

4.1.2 Refined Classification of Roles in Community

As comprehensive as it seems to be, the above role classification by Nakakoji et al. (2002) still has some shortcomings. First, the class of Reader is not quite meaningful because Readers do not contribute to the community directly. The above definition of Reader simply states that Readers only read codes. Reading is useful to the Readers but not to the community unless Readers contribute in other ways. Nakakoji et al. explained that Readers were like peer reviewers. This analogy is not appropriate. Peer reviewers must pinpoint the weaknesses of the system reviewed. If Readers also pinpoint the weaknesses, they become Bug Reporters. So the type of Reader is transient. A reader must become another type of member in order to contribute back to the project what she learnt from reading. If she does not contribute, a Reader is not different from a Passive User. The position of Reader in the hierarchy of the above classification is also awkward

because the class Bug Reporter is higher than Reader but the former does not necessarily read codes. Thus the class of Reader should be eliminated since it adds no value. Actually, this class was not used in other previous studies.

Second, the class of Bug Reporter should be extended to include both Bug Reporter and Feature Requestor. Or better, it should be called Active User, in contrast to Passive User. In addition to reporting bugs, Active Users can contribute to the project by requesting new features, even though they may not understand programming at all. From these requests, the developers know what the actual users really want. The Active Users may as well exchange other information such as social events by posting messages to communication tools like mailing lists and forums. This kind of information exchange was not considered by Nakakoji et al. (2002).

Third, the class of Bug Fixer is an outlier. The explanation "Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs" is not to the point of real debugging practice. Knowing where the cause of the bug lies is the premise of solving the bug. In some cases, the Bug Fixer needs to be familiar with the structure of the whole system and the interaction among many modules in order to identify the location or locations of the cause of a bug. That means reading lots of codes and understanding a big portion of the system. So the amount of codes read by developers is not a good criterion for classifying roles. More importantly, the definition of Bug Fixer is not consistent with the definitions of other developer classes. The former is based on the content of action – debugging. The latter is based on the frequency of action, from Peripheral Developers who act irregularly to Core Members who act continuously. Furthermore, the Peripheral Developers were assumed by Nakakoji et al. as only adding

new functions or features, not dealing with bugs at all. In reality, Bug Fixers may act frequently or irregularly. And they may also add new functions. Similarly, people who add new functions may also fix bugs. So it is better to use the single consistent criterion – frequency of contribution – to define developer classes. The class of Bug Fixer can be eliminated.

Fourth, the class of Peripheral Developer should be mended. As explained above, Peripheral Developer can also fix bugs.

Fifth, the class of Active Developers should be renamed as Central Developers. The use of "active" may lead to a false deduction that other categories of developers would be inactive or passive. The use of "central" is more consistent with the metaphor of "from peripheral to central to core".

Sixth, the class of Core Members should be renamed as Core Developers, to be consistent with other developer categories.

Seventh, the class of Project Leader assumes that the leader is only one person. This is not true for those projects which have a few people acting as leaders. One good example is the Apache project (Mockus et al. 2000).

Eighth, other important contributions that were ignored by Nakakoji et al. are support and documentation. The criteria of frequency of action can also apply to contributions other than bug fix and feature addition. So support, documentation, and information exchange should be added to the definition of developers including Peripheral Developer, Central Developer, Core Developer, and Project Leader.

As a result of the above critics, the classification of roles played by OSS community members can be refined to consist of six roles: Passive User, Active User, Peripheral Developer, Central Developer, Core Developer, and Project Leader. These six roles can be grouped to two categories: User Group for Passive User and Active User, and Developer Group for the other roles. The summary of these roles are delineated in Table 2. As indicated by Nakakoji et al. (2002), not all projects have all these types of members. And the role assumed by a member may change as her contribution changes.

Table 2: Roles played by members in OSS communities

Role	Contribution
User Group	
Passive User	No direct contribution other than forming a larger user base.
Active User	Report bugs, suggest features, and exchange other information.
Developer Group	
Peripheral Developer	Irregularly fix bugs, add features, provide support, write documents, and exchange other information.
Central Developer	Regularly fix bugs, add features, provide support, write documents, and exchange other information.
Core Developer	Extensive personal contribution (including all kinds of contributions of Central Developer), coordination of other developers.
Project Leader	Significant personal contribution (including all kinds of contributions of Core Developer), management of the project.

4.2 OSS Methodology

It is the methodology that distinguishes an OSS project from other kinds of closed source software projects. The methodology adopted by most OSS projects is generally referred to as the OSS development model (Feller & Fitzgerald, 2000). It is a novel

domain that is not covered by existing theories such as Project Management and Software Engineering. This thesis does not discuss the OSS development model in detail because this topic along is worth a dedicated work and there are already some studies published. However, this thesis inspects an interesting topic: OSS development model and the software crisis.

The chronic symptoms of the software crisis have been well-known since the 1960's. A nontrivial portion (at least one in four) of Information System projects carried out in the traditional software engineering way even failed before delivering a stable system (Keil et al., 2000). For those eventually delivered, they usually took much longer than the scheduled time to develop, the cost of development was much higher than budgeted, and the quality of the delivered systems were usually inferior to the original specifications (Feller & Fitzgerald, 2000; Keil et al., 2000). Feller and Fitzgerald (2000) argued that OSS development model was promising in addressing these three stubborn diseases of the software crisis namely: speed, cost, and quality. But there is not yet an academic study that provides empirical evidences on comparison of the two approaches in system development. The following sections conduct an exploratory study of the development speed, development cost, and development quality of OSS.

4.2.1 Development Speed

So far, most studies of software evolution have been performed on proprietary systems developed within the boundary of a single company abiding by the traditional software engineering model (Godfrey & Tu, 2001). Results from these studies suggest that the growth rate slows down as the system gets larger and more complex (Godfrey &

Tu, 2000). And one classical assertion (called Brooks's Law) about development speed of traditional software is: "Adding manpower to a late project makes it later" (Brooks 1995).

OSS brings a new research area for software evolution. OSS is developed by voluntary developers and users scattered all over the world, who cooperate via communication tools without the management mechanism and overhead of a commercial enterprise. It has been demonstrated that some big OSS projects such as Linux and Apache evolved at super-linear speed (Godfrey & Tu, 2000; Mockus et al. 2000). After examining the growth of a few other OSS projects like VIM, Fetchmail, and GCC, Godfrey and Tu (2001) found that each system told a different story.

However, no research quantified the development speed across a large sample of OSS projects. Furthermore, there are no studies that empirically identify the factors that may impact the development speed of OSS projects, let along carried out investigation of the interaction among the factors. This thesis suggests some promising research directions.

"Release early, release often" is promoted by Raymond (2001) and agreed by other OSS researchers (Feller & Fitzgerald, 2000) as one of the distinguishing characteristics of OSS. The frequent, incremental releases demonstrate vividly the rapid development and evolution of OSS.

Abiding by "release early", many OSS projects started to put their releases on line since their inception. Conventionally, the publication of the 1.0 release can be treated as the milestone for a project to reach stable stage. In some cases, the first publicly available stable release (not alpha or beta version) has a version number greater than 1.0. For OSS projects, there could be tens of 0.x public releases before the stable 1.0 release. One aim

of these pre-1.0 versions is to build up the development community (Robbins 2002). This case of many public pre-1.0 versions can rarely be seen before in closed source software. Though some closed source software now follow this strategy in practice, they cannot gain the same benefit from early releases as OSS. Because their source codes are not published, they cannot have the "eyeballs" from the community to diagnose the codes.

However, one caveat for "release early" is that the earliest release should be workable, though it may be in primitive shape with bare minimum functions. "Show me the code" is a famous statement from Linus, the originator of Linux. It shows the emphasis of the OSS community on codes, the brain child of programmers. Having a working executable version is a prerequisite, touted by Raymond (2001), to the building of a community around an OSS project. A famous example of this caveat is the initial failure of Netscape's breakthrough action in 1998 to make its browser Mozilla open source (DiBona et al., 1999). Because the codes released by Netscape were not in an executable state, the interests of community on Mozilla faded quickly. It was until 2002, four years after the initial release, that the 1.0 version of Mozilla finally came out.

Abiding by "release often", the frequency of releases in an OSS project can be as high as several releases per day, as shown in the early days of Linux (Raymond 2001) and in the FreeBSD project (Jorgensen 2001). Usually, the interval between releases is in terms of months, weeks, or days. On the contrary, proprietary software often has much longer release cycles. For example, popular Microsoft software such as Money has a release interval of one year.

The distinction of development release and stable release is one good strategy that is adopted widely in OSS projects, leading by the Linux project. Development releases are

aimed for developers and power users who are not scared by possible bugs brought by new features. Cutting edge new functions are only incorporated into the development releases. On the contrary, stable releases are for ordinary users who want to have a relatively bug-free version, though this version may lack some new features. Only tried and true features in the development release can be brought into the stable release after thorough tests. As a result, the stable releases may lag behind the development releases in terms of features. However, the stable releases are less risky than the development releases.

It is interesting to note that the highly incremental model for OSS may not support well the development of complex new features, such as the Symmetric Multiprocessing subproject of the FreeBSD project (Jorgensen 2001). The too frequent releases on the development branch became a burden for the subproject to catch up with.

In a word, examining the earliness and frequency of releases is a good way to inspect the development speed of OSS projects. Another measurement of development speed can be the length of time that an OSS project took to reach certain development status, such as the stable or mature stages. Relevant data about releases and development status can easily be obtained from the public archives of OSS projects.

4.2.2 Development Cost

For the development cost issue, Feller & Fizegerald (2000, p. 64) argued that OSS programs can be downloaded freely or be purchased with only nominal fee. In this way, they made a logic error by substituting the concept of "development cost" with "purchase price". If there is no existing software that can meet the requirements, development cost is inevitable for building a new application no matter under which development model.

Feller & Fizegerald (2000) made a good point by citing the fact that maintenance of software was very costly. But they treated the cost of maintenance as a part of the total cost of software development. In my opinion, maintenance cost should be a consistent part of Total Cost of Ownership, of which the development cost could be an optional part. The reason can be seen clearly in the case of adopting an existing software application rather than developing a new one. In this case, there is virtually no development cost at all, though there is still maintenance cost.

The problem is, then, how to measure the development cost in the OSS development model. There is no central record of material cost, since OSS developers and users use their own equipments and Internet connections. At most, we may calculate the cost of the portals, like the comprehensive services provided by SourceForge.net. The cost of manpower is even harder to measure since most OSS developers work voluntarily in their own paces. It is unrealistic to ask every developer to record how many hours she put on a specific project, let alone to assign an appropriate hourly rate for each of them. So realistically, the development cost in the OSS development model can only be roughly estimated.

The on-line motivation study on Linux (http: //www. psychologie. uni-kiel.de /linux-study) is probably the best known source where developers were asked to state how many hours they spent per week on Linux development. The current results from this research are a mean of 18.4 and a range of 1-70 (Hertel et al., 2002). And a previous estimate of 13.9 hours per week was used as a proxy by Koch and Schneider (2002) for calculating working time for the GNOME project. It is easy to understand the difficulty faced by Koch and Schneider when accurate measures of time are unavailable for the GNOME

project. There are other similar estimates gathered from surveys in the FLOSS project (Berlecon Research, 2002).

A roundabout way of effort and cost estimation for OSS project was piloted by Wheeler (2002a) and followed by Gonzalez-Barahona et al. (2002). They calculated the total of lines of code for an OSS project. Then the formulae and assumption of the COCOMO cost and effort estimation model (Boehm 1981) were used to calculate the estimated development effort in person-years, estimated schedule in years, and estimated cost to develop. Basically, the OSS project was estimated as though it was developed in the traditional proprietary development model. As they admitted, there are vast differences between the OSS development model and the traditional proprietary development model. So their approach sheds no light at all on how much the real development cost and effort would be for the OSS project under OSS development model. We only know how much it would cost if the project is under traditional model; we do not know how much we would save by adopting the OSS development model.

Hence, further research is needed on how to measure the development cost of OSS projects. The success of this kind of research may rely on the cooperation of OSS community. For example, if the developers can record their time spent on a project, this can facilitate the researchers a lot. Right now, one major deficiency of the public archives of OSS projects is the lack of cost-related data. This situation may improve when the OSS community recognizes the importance of recording cost-related data.

4.2.3 Development Quality

The difficulty of assessing the quality of software can be seen from the statement: "It is extremely difficult to evaluate the quality of the software developed by FreeBSD to a

more accurate level than simply stating that the operating system is working and has many users" (Jorgensen 2001, p. 325).

Many authors adopted a pragmatic way to assert that OSS's superiority in quality was proven by the significant market share occupied by some OSS products without any commercial marketing campaigns (Chandra & Chen, 2000; Feller & Fitzgerald, 2000). So Apache with its 65.39% market share in active sites as of October 2002 (http: //www.netcraft.com /survey) should be inarguably better than its competitors like Microsoft IIS. To some extent, their ideas are valid. However, market share is only one possible indicator of software quality if we can believe the rationality of the market. Sometimes, a dominant market share is gained due to factors other than software quality, such as marketing power (e.g., Microsoft excels in this tactic), legacy reason, or political reason (e.g., a policy to use only domestic products). Generally, OSS products have not been widely adopted in the business world due to many reasons. Accordingly, many OSS products may be deemed to have inferior quality if market share is the criterion. This is simply not rational.

Other arguments of the high quality of OSS include the high quality of OSS developers who are assumed to be the most-talented and highly motivated 5% of software developers (Raymond 2001), and the independence of peer review (Feller & Fitzgerald, 2000). These are just possible causes of high quality, not necessarily an insurance of high quality, nor indicators of high quality.

More quantified quality indicators are needed to evaluate the quality issue more comprehensively. Previous research on software quality classifies it into internal software quality, which is the quality from the point of view of the creator of the software, and

external quality, which is the quality from the point of view of the users of the software (Anderson et al., 2002). Hankin et al. (1996) indicated that the notion of quality is multifaceted but must include programmer productivity, verifiability, reliability, maintainability, and efficiency.

Importantly, the public archives of OSS projects can be the supplier of data on quality. Thus, investigating the quality of OSS based on data from the public archives of OSS projects is a viable approach. There are already pilot studies adopting this approach. For instance, Mockus et al. (2000) checked the defect density of the Apache project. And Stamelos et al. (2002) examined code quality of 100 C applications. This thesis contributes exploratory ideas along this approach by proposing an assessment framework for OSS projects in a later section.

4.3 OSS Products

Like in closed source software projects, the major objective of an OSS project is to produce useful software products. What differentiates the OSS products produced by an OSS project from products produced by closed source software projects is, of course, the feature of open source.

OSS products such as the Linux kernel and the Apache server are examined frequently in previous studies. This thesis explores two areas that are not covered in reviewed OSS research: the impacts of programming languages and the documentary structure.

4.3.1 Impacts of Programming Languages

Programming languages are as important to software projects as natural languages to social life. Unfortunately, they did not draw much serious attention in previous OSS studies. The C programming language is the dominant language used in OSS projects (Wheeler 2002a; Gonzalez-Barahona et al., 2002), especially in those high-profile projects examined in previous OSS studies. As a result, the C language is the only language studied (implicitly or explicitly) in most previous OSS studies.

Wheeler (2002a) and Gonzalez-Barahona et al. (2002) are exceptions where the sizes of programs in other programming languages were also measured. However, the authors did not differentiate the nuances among those languages. They virtually treated all programming languages as equal. For example, they simply summed up the total of lines of codes for all programs, no matter in which languages these programs were written, in their estimation of development cost. This kind of ignorance of differences among languages is based on an old rule of thumb (Prechelt 2000), which says that programmer productivity measured in lines of code per hour is roughly independent of the programming language. This rule of thumb is also an explicit assumption of some widely used effort estimation model such as COCOMO (Boehm 1981).

However, 100 lines of code in lower level language like C are not as powerful as 100 lines of code in higher level language like Python. If lines of code are used as an indicator of project complexity, the differences between programming languages should be taken into account. Accordingly, the 100 lines of code in Python should imply a higher complexity than the 100 lines of code in C.

Programming languages are not created equal. Each one has its own idiosyncrasies and suitable application areas. The ignorance of differences among programming languages and the narrow focus on the C language are two shortcomings of previous OSS research, though the dominance of C language in OSS projects can be an alibi for the narrow focus. Furthermore, the impacts of programming languages on software projects are not within the theme areas and strategic directions in research on programming languages identified by Hankin et al. (1996). Those theme areas and strategic directions focus on programming languages per se, not on the relationship between programming languages and software projects. The ignorance of this relationship is still reflected in very recent studies (e.g., Pepper et al., 2002) on programming languages.

This thesis makes an initial attempt to delineate the impacts of programming language on software projects. Specifically, programming languages may exert impacts on design philosophy, programming style, specific application area, development speed, development tools, libraries and components, source code size, expertise, and other subtle issues.

- Each programming language has its underpinned design philosophy. For example, object-oriented languages such as C++, Java, and Python are fundamentally different from those structural languages such as C and Pascal.
- Each programming language also implies or encourages certain programming style. For instance, the mantra of Perl is "There's more than one way to do it" (Holden 2002). As a result, programs in Perl are notoriously hard to understand and maintain (Chun 2001). On the contrary, the motto of Python is "There should be one – and

preferably only one – obvious way to do it". Thus, programs in Python are much easier to understand and maintain (Brown 2001; Chun 2001).

- Each programming language has its own specific application area. Although some languages like C and Python may be versatile enough to be applied in almost any area, many languages are targeted at and are fine tuned for certain specific application areas. For instance, JavaScript is for client side web scripting. FORTRAN is for scientific calculation. And COBOL is for business application.

- Development speed can be determined to a great extent by the nature of programming language. For example, high level scripting languages like Python can be 7 times faster to develop a prototype of a system than low level system languages like C (Brown 2001 p.7). Programming time in Java could be three or four times as long as programming time in Python (Holden 2002 p. 12). In an empirical comparison of seven programming languages for solving an identical problem, Prechelt (2000) found that scripts (Perl, Python, Rexx, Tcl) took less than half as long to write as programs in conventional languages (C, C++, Java).

- Programming tools, like the Integrated Development Environment, that support certain programming languages are another facilitator of development speed. For example, though relatively a new language, Java quickly became one of the most popular languages. Apart from its native features by design, its large set of tools contributes a lot to its prevalence.

- Each successful programming language is backed up by its rich libraries and components. FORTRAN, C, and Perl are just three famous instances. Exampled by

Python and Jython, the smart strategy of seamlessly integrating modules in other languages could significantly extend the richness of libraries and components.

- The size of source codes may vary significantly for different languages. With native support for advanced data structure such as list and dictionary, high level scripting languages like Python and Perl tend to produce smaller size of codes compared to conventional languages like C and Java, in which only basic data structure is supported. Python code is typically 3-5 times shorter than equivalent Java code and often 5-10 times shorter than equivalent C++ code (Holden 2002 p. 12). In his same research, Prechelt (2000) found that programs written in conventional languages (C, C++, Java) were two to three times longer than scripts (Perl, Python, Rexx, Tcl).
- Just like different natural languages have different population of speakers, different programming languages have different population of experts. One peculiar example is that the reason why some security bugs in the GNU Mailman project lurked for 3 years without being discovered could partly due to the relatively lower popularity of the programming language – Python (Payne 2002). There were simply not enough qualified "eyeballs" for Python, compared to more popular languages like C.
- There are other subtle impacts of programming language. For example, license issues, personal tastes, legacy systems, etc.

Because a programming language can really make a difference, it is not wise to compare, without discrimination, programs written in different programming languages. To make things more complicated, many OSS projects are written in more than one programming languages. For instance, it is a kind of best practice for a program in

Python to have its critical modules rewritten in C in order to improve the execution speed.

This can bring a practical problem for OSS researchers. Even if an OSS researcher has the luxury of using commercial tools such as Logiscope used by Stamelos et al. (2002), the tool may not support the specific programming language. Anyway, the workload will definitely increase when more programming languages are involved.

Many programming languages are used in OSS projects. For example, there are more than 40 languages deployed in projects hosted at SourceForge.net (http: //sourceforge.net). This richness of programming languages is actually a bless to OSS researchers. The same kind of application may be implemented in different programming languages by various projects, in the same language by different projects, and in different languages by the same project. For example, email clients are written in Python, Java, PHP, C++, Delphi, C, Visual Basic, etc. Researchers can find ample opportunities to conduct advanced study on the impacts of programming languages on real projects. Research like Prechelt (2000) can be extended to study non-trivial applications in different languages in real world settings.

To study the same kind of application in different languages is a good approach to studying the impacts of programming languages on software projects. Another approach is to study projects which use a certain programming language as a group, and results from different groups classified by programming languages can be compared and contrasted. One appealing advantage of this approach is that the real world usage of programming languages can be fathomed comprehensively.

Python is a promising language promoted by many OSS figures including Eric Raymond (2000). It has many features and advantages like being free since it is OSS, very high level language, object-oriented, portable, extensible, scalable, powerful, robust, effective as Rapid Prototyping tool, easy to learn, easy to read, and easy to maintain (Brown 2001; Chun 2001; Lutz 2001). It is good at rapid application development, cross-platform development, component integration, Internet programming, database programming, distributed programming, system utilities, graphical user interface, text processing, mathematics, and many other applications (Brown 2001; Chun 2001; Lutz 2001).

Although there are 1907 projects at SourceForge.net having Python as one of their programming languages as of 3 October 2002, projects in Python have not been analyzed in previous studies. This thesis fills this gap by choosing projects in Python as the major subject of an empirical study in Section 7. It certainly ameliorates the narrow focus on C language in previous studies by contributing new knowledge about Python projects to the school of OSS research.

4.3.2 Documentary Structure

In an astute paper, Van De Vanter (2002) studied what he called "documentary structure" – textual aspects added by programmers for the sole purpose of aiding the human reader but explicitly defined to be not part of a programming language. These aspects include indentation, inter-token spacing, line breaks, comments, and the choice of names for language entities. He stated that though largely ignored in the research literature, documentary structure occupied a central role in the practice of programming.

Unfortunately, the only documentary structure studied in previous OSS research is comments. Comments can facilitate the understanding of source codes. It is a general rule of good programming to add adequate and appropriate comments to programming codes while writing or updating a program. The codes show "how" – how to complete a task, while the comments show "why" – why the task should be completed this way. So good comments should explain the reasons behind the codes, rather than simply translate the codes into natural language.

The quality of comments is hard to measure without manually assessing the comments. However, the quantity of comments can easily be measured automatically by tools. Comment frequency, the proportion of comments to executable statements, is used to measure the self-descriptiveness of software modules, which is one criterion taken from a software quality standard by International Standards Organization (Stamelos et al., 2002). In an OSS project, a program may be updated by more than one developers scattered worldwide (Mockus et al., 2000). So self-descriptiveness should be an important requirement for open source code (Stamelos et al., 2002). Actually, coupled with other metrics such as reliability and bugs, knowing the percentage of comments in real projects can give empirical evidence of how much effort should be put into comments.

Although many previous studies differentiated comments from codes when calculating lines of code, not all of them cared about the percentage of comments in the archives. Most of the results of comment frequency from previous studies are related to C programs. Stamelos et al. (2002) found that the average comment frequency for 100 Linux applications was too low – only 11%. Godfrey and Tu (2000) considered that the

percentage of comments and blanks for the Linux kernel was at healthy level – between 28% and 30%. Analysis of other languages was done in a study (Prechelt 2000) not related to OSS, in which programs written in conventional languages (C, C++, Java) had a comment frequency of 22% while scripts (Perl, Python, Rexx, Tcl) had a comment frequency of 34%.

There are special forms of comments that should be handled differently. One form is the in-line comments, which can start in the same line of code after a special identifier (e.g., "#" in Python, "//" in C / C++ / Java). Thus, simply counting the lines of comments is not enough to give an accurate measurement of comments. Previous studies on comments did not explain how they dealt with in-line comments.

Another form is like the "docstring" in Python. It can act as comments in the source code. However, it will be complied into the binary code and act as help information during program execution. So it should be treated as code (actually data) rather than comments. It is not even mentioned in previous studies on comments.

Interestingly, one great feature of Python is that indentation is used as delimiter of programming structure. This is really a smart idea since programmers have been using indentation as visual indication of program structure. So among Python programmers, there is no waste of time and effort in arguments about where to put the "{" and "}" as in other languages such as C. And there is no worry about mismatch between the visual structure indicated by indentation and the syntactic structure stipulated by delimiters such as "{" and "}". This kind of mismatch is often a cause of bugs in other languages. In Python programs, what you see is always what you get: the visual structure is exactly the syntactic structure. This feature is a vivid example of Python's motto – "There should be

one – and preferably only one – obvious way to do it". Van De Vanter (2002) did not recognize the existence of this feature.

Evidently, the public archives of OSS projects are an unrivaled resource for research on Documentary Structure.

4.4 OSS Services

Services provided by an OSS project, actually by the OSS community, include support, documentation, training, education, consultation, socialization, etc. Because OSS products can be obtained by anyone freely, providing services is the only viable way to make profits from an OSS project. This leads to an important topic in OSS research: business models based on OSS. This topic faces difficulties on both academic side and practice side. On the academic side, there is a scarcity of academic research on this topic. Relevant studies found in reviewing the literature for this thesis are mostly either working papers (e.g., Hecker 2002) or narrations (e.g., http: //www.opensource.org /advocacy /case_for_business.php) published in the Internet. On the practice side, many start-up companies based on OSS failed (Berlecon Research 2002) with the burst of recent dot com bubbles.

Nevertheless, business models based on OSS are paramount to the vitality of OSS movement. Researchers have already made some good points. For instance, Ousterhout (1999) indicated that profitable company can be established based on OSS since open source and commercial developments are symbiotic. And it does not require much start-up fund to establish or to expand company based on OSS (Sandred 2001, p. 156). Moreover, there are already some models summarized from the practice. The most famous models are those proposed by the Open Source Initiative (http:

//www.opensource.org /advocacy /case_for_business.php): support sellers, loss leader, widget frosting, and accessorizing. On top of these models, Hecker (2002) proposed more models as quoted below.

"Support Sellers," in which revenue comes from media distribution, branding, training, consulting, custom development, and post-sales support instead of traditional software licensing fees.

"Loss Leader," where a no-charge open-source product is used as a loss leader for traditional commercial software.

"Widget Frosting," for companies that are in business primarily to sell hardware but which use the open-source model for enabling software such as driver and interface code.

"Accessorizing," for companies which distribute books, computer hardware and other physical items associated with and supportive of open-source software.

"Service Enabler," where open-source software is created and distributed primarily to support access to revenue-generating on-line services.

"Brand Licensing," in which a company charges other companies for the right to use its brand names and trademarks in creating derivative products.

"Sell It, Free It," where a company's software products start out their product life cycle as traditional commercial products and then are continually converted to open-source products when appropriate.

"Software Franchising," a combination of several of the preceding models (in particular "Brand Licensing" and "Support Sellers") in which a company authorizes others to use its brand names and trademarks in creating associated organizations doing custom software development in particular geographic areas or vertical markets, and supplies franchises with training and related services in exchange for franchise fees of some sort.

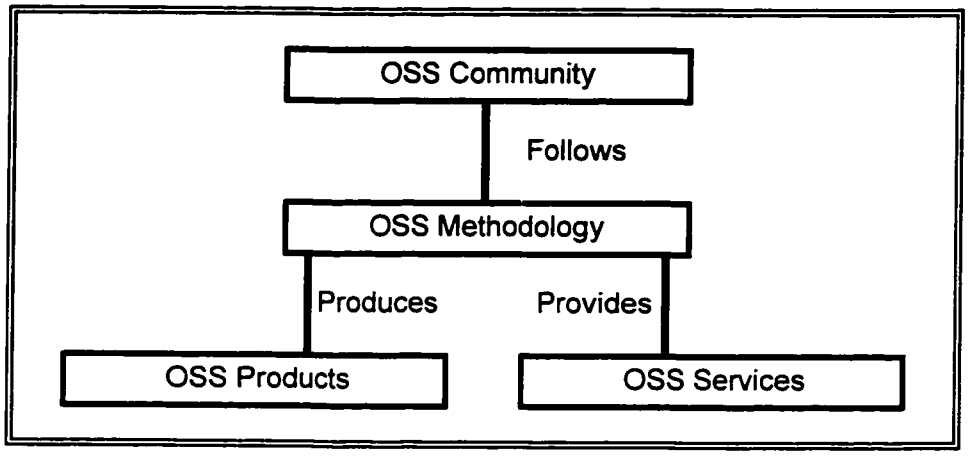
The worth of these business models is still to be thoroughly tested in real world endeavors. However, other kinds of services like support, documentation, and socialization are reflected in the public archives of OSS projects. For example, forums and mailing lists are often the resources for in time support.

4.5 Summary

Section 4 proposed an initial definition of an OSS project. An OSS project has four critical components: community, methodology, products, and services. The OSS community follows the OSS methodology to produce OSS products and to provide OSS services. Figure 3 shows a conceptual diagram of OSS Projects, one of the major entities studied in this thesis.

Major topics discussed in Section 4 encompass the specification of roles played by OSS community members, OSS development model and software crisis, impacts of programming languages, documentary structure of source code, and business models based on OSS. Research on all these topics can benefit from the public archives of OSS projects.

Figure 3: A conceptual diagram for OSS projects



5 Strategic Level Study of OSS Adoption

Although OSS is burgeoning, it has not conquered the world. On one hand, many people have benefit a lot by embracing OSS. For example, Davis et al. (2000) described a vivid case in which OSS products furnished more capable, more reliable, more efficient, more supportive, and much less expensive services to a university. The Floss project (Berlecon Research 2002) provides further evidence that OSS has been widely adopted in Europe. On the other hand, there are perhaps more people who are still skeptical of the ideology of OSS as well as the superiority of OSS products. As a result, OSS is still a challenger to dominant proprietary closed source software in most application areas, with a few well-known exceptions such as Apache (www.apache.org) and BIND (http: //www.isc.org /products /BIND/).

But from the viewpoint of marketing, being a challenger is not necessarily a bad situation for OSS. It really means that OSS still has vast potential of development. The theory of Innovation Diffusion (Rogers 1995) distinguishes six kinds of adopters: Innovators, Early Adopters, Early Majority, Late Majority, Laggards, and Rejecters. Prevailing in the literature, this classification of adopters is based on the criterion of timeliness of adoption. Most OSS products now probably just attract the attention of Innovators and Early Adopters. All those skeptical people are still potential OSS adopters who may become Early Majority, Late Majority, or Laggards. They need a decision support model to help them recognize the benefits of OSS, evaluate OSS products and services, and make decision on OSS adoption. Hence, the study of OSS adoption is practically very helpful for the promotion of OSS among these potential adopters.

Although the study of Information Technology (IT) Adoption is abundant in the literature, there is a lack of published research on the adoption of OSS. In the reviewed literature, Wang & Wang (2001) is the only published academic study on OSS adoption model. Thus, the study of OSS adoption can make theoretical contributions as well. After pinpointing the deficiency of existing adoption models, this section introduces a novel approach to the study of OSS adoption.

5.1 Levels of Study of OSS Adoption

The Diffusion of Innovations (Rogers 1995) model and the Technology Acceptance Model (Davis 1989) are two prevailing theoretical models in the research of IT adoption. The former identified five perceived attributes of an innovation (relative advantage, complexity, compatibility, trialability, and observability) that can influence the adoption behavior. The latter focused on two such attributes (usefulness and ease-of-use). Numerous studies (e.g., Gallivan 2001; Gefen 2000; Jackson et al., 1997; Karahanna et al., 1999; Szajna 1994; Thong 1999) are based on these two models or their variants.

Although locus of adoption – individual vs. organization – was identified in the literature, traditional adoption frameworks like the above two focus on individual user as the level of analysis (Gallivan 2001). The literature also recognizes the importance of tailoring adoption theories to the adoption context (Thong 1999). Those attributes of an innovation used in the Diffusion of Innovations model and the Technology Acceptance Model are only one element of the adoption context. In addition to the characteristics of innovation, the literature identified three other elements of the adoption context: characteristics of the organizational decision makers, characteristics of the organization,

and characteristics of the environment in which the organization operates (Thong 1999). Obviously, the adoption context is studied at the organization level.

However, the literature on technology adoption and innovation diffusion has one limitation. It assumes that the usage of the innovation or technology is the same for all potential adopters. This may be true to some extent for software, since tactically all software tools have their designed set of functions with a norm of usage. For instance, word processing software is for document editing. However, if we think strategically, the same tool can be used in quite different ways. One good example from prominent innovations is the usage of gunpowder. It can be used to kill people; it can also be used to help people. It is the adopters who decide the different usages of the same innovation, as in the saying "It is the man with a gun who kills, not the gun that kills".

In reality, not all adopters utilize the adopted innovation in the same way. This issue is salient in the context of OSS. The purpose of adoption of OSS for Microsoft is definitely different from that for an OSS developer. The former has an infamous intention of "embracing, extending, and extinguishing" (Raymond 2001). The latter has an altruistic intention of "embracing, contributing, and prospering". Thus, the purpose of adoption is important when the technology or innovation can be used in different ways. And the adopters can be distinguished by their purposes of adoption.

In previous studies which abide by the two dominant models, the adoption of an IT was studied as the question of "the extent of willingness to use the IT or not" – a "whether" question. That's sufficient for these studies since their subjects would only use the specific IT in the same single way. They did not consider the question "how to use the

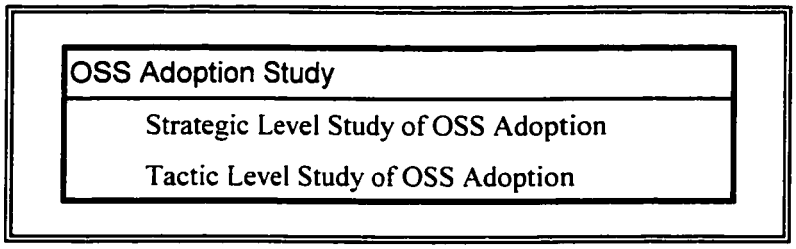
IT in different ways". Furthermore, all the software tools studied before are proprietary software. This also constrains the possibility of more different usages of the software.

The availability of source codes and the freedom of modifying and distributing source codes in OSS provide the prospect of using OSS in quite different ways. For example, OSS can be deployed as a source of revenue and profit for an enterprise like Red Hat (distributor of GNU/Linux software) and O'Reilly (publisher of OSS books). The question "how to adopt OSS" should be considered before the question "whether to adopt OSS" since the answer to the former is the key to the latter. For example, it is reasonable to say "We should adopt OSS because it can be used in such an innovative way that will bring values, profits, and competitive advantage to us". Hence, the former is more important than the latter.

Accordingly, OSS adoption can be studied from a strategic perspective, by considering the "how to adopt" question before answering the "whether to adopt" question. The "how to adopt" question is not about the procedure of adoption or the sequence of actions (e.g., initiation, adoption, and implementation) as studied in some previous research (e.g., Thong 1999). Instead, it concerns the various ways in which the same OSS can be deployed by different adopters. In this sense, this question is never asked in previous IT adoption studies where it was assumed that there was only one uniform way of using the candidate IT. To emphasize its strategic intent, it may be better to rephrase this question as: how to strategically utilize the OSS in different ways? This strategic level of study should bring more insights than the tactic level in previous adoption studies which was mainly concerned with the "whether to adopt" question.

This thesis suggests a novel approach for the study of OSS adoption. The study should be carried out at two different levels: strategic level and tactic level. The strategic level study of OSS adoption aims to answer the question: how to strategically utilize OSS in different ways? It identifies the viable Strategic Usages of OSS and classifies adopters according to their major Strategic Usage of OSS. It is only after the strategic level study that the tactic level study of OSS adoption goes into action. The tactic level study of OSS adoption aims to answer the question: how to tactically select appropriate OSS for a determined Strategic Usage? Previous research simply omitted the strategic level study of IT adoption by assuming a uniform usage of an IT, and dived directly into the tactic level study by examining whether a particular adopter would like to put the IT into a predefined usage. Figure 4 shows the proposed two levels of OSS adoption study.

Figure 4: Two levels of OSS adoption study



Actually, the word "adoption" is more in line with "acceptance". So Davis (1989) made a wise choice when he selected "acceptance" rather than "adoption" in the title of his theory – the Technology Acceptance Model. It is then easy to understand why the previous study on IT adoption focused on the acceptance of IT rather than the actual usage of IT. In my opinion, the word "utilization" has more proactive connotation than "adoption", which is more passive. And utilization covers the domain of adoption since acceptance is a premise of utilization. The term of "OSS Utilization" is then more

appropriate to emphasize the different actual usages of OSS than the term of "OSS Adoption". Thus, the theoretical discussions in Section 5 and 6 are really about OSS utilization, not only about OSS adoption. However, since adoption is a common term adopted by the literature, the term of "OSS Adoption" is still used in this thesis in stead of "OSS Utilization". Similarly, the generally accepted term of "Adopter" is still used in lieu of the more suitable term of "User".

In the following sections, the strategic level study of OSS adoption is conducted by defining the contents of OSS adoption, introducing two novel perspectives to the study of OSS adoption, identifying the Strategic Usages of OSS, and classifying OSS adopters according to their major Strategic Usage of OSS. The tactic level study of OSS adoption is fathomed in Section 6.

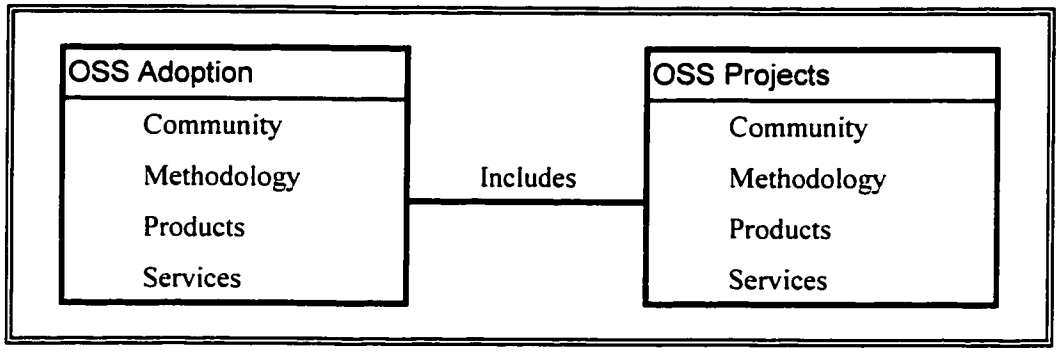
5.2 Contents of OSS Adoption

In previous research on IT adoption, the contents of adoption are basically just the software products. This is another limitation of previous research. As described in Section 4, an OSS project consists of four components: OSS community, OSS methodology, OSS products, and OSS services. The contents of OSS adoption should be much richer than the adoption of OSS products. Actually, it should include the adoption of all the four components of an OSS project. Figure 5 outlines the relationship between the contents of OSS adoption and the components of an OSS projects.

The contents of OSS adoption should include the adoption of OSS community. Obviously, when an adopter uses an OSS product, she automatically becomes a member of the associated OSS community. She is at least a Passive User of the community. And she may play other roles including the various developer roles as discussed in Section 4.1

if she wants to contribute more to the community. In the case of closed source software, however, an adopter can hardly play a developer role.

Figure 5: Contents of OSS Adoption



The contents of OSS adoption should include the adoption of OSS methodology. The adoption of OSS methodology can be in two forms: passive and active. In the passive form, an adopter of OSS may not agree to the methodology. But she has to abide by the rules stipulated by the OSS methodology since it is the way by which the OSS community develop the OSS products and provide services. In the active form, an adopter of OSS heartily supports the OSS methodology.

One special case of active adoption of OSS methodology is to apply the OSS development model to the production of new OSS products and services, or even non-OSS products. Here, the OSS methodology is the major target of OSS adoption. And new community, products, and services will be built up along the process. Dinkelacker and Garg (2001) discussed the application of OSS methodology within the boundary of an organization. In this situation, the OSS community is constrained to only members of the organization, not the general public in the case of conventional OSS projects. Source codes and design documents are only open to this limited-scope community. Then non-OSS products can be produced in this kind of limited-scope OSS development model.

The contents of OSS adoption should include the adoption of OSS services. As a member of the OSS community, an OSS adopter can enjoy the plentiful services provided by the community. Many of these services, such as support provided via mailing lists and forums, are free. Conversely, most services provided by proprietary software vendor are associated with a price tag. It is unwise not to take advantage of OSS services in OSS adoption. Furthermore, an adopter can also contribute back to the community by providing various services. As discussed in Section 4.4, supplying value-added services is the gist of many business models based on OSS.

Hence, it is appropriate to consider OSS adoption as the adoption of the OSS project, including the adoption of OSS community, OSS methodology, OSS products, and OSS services. This project-based adoption approach is superior to the traditional product-based adoption approach used in previous IT adoption research.

5.3 Novel Perspectives

Not constrained by the traditional perspective of timeliness of adoption, this thesis introduces two novel perspectives to the study of OSS adoption. One is the Orientation perspective, which concerns the purpose of OSS adoption. Another is the Participation perspective, which concerns the interaction between adopters and OSS communities.

5.3.1 Orientation

As described above, the purpose of adoption is a salient issue in the context of OSS. From the perspective of orientation, the purpose of OSS adoption can be classified into two coarse-grained categories: value-oriented and profit-oriented.

Value orientation is based on the innate values, or "use value" in terms of economics, of the OSS: its functionality, its quality, its cost-effectiveness, its strong community, etc. Under this orientation, the adopter is mainly interested in exploiting an OSS by directly consuming these innate values. In another word, the use value of the OSS can only be realized in actual usage. Different from wearable commodities, the more usage a software obtains, the more use value it retains. Thus, value orientation inclines to increase the use value of the OSS. Generally, under value orientation, the adopter does not treat the OSS as a source for generating profits. However, the adoption of OSS and the usage of the use value of the OSS can often reduce the cost of the adopter and add more values to the adopter, like the example in Davis et al. (2000). The reduction of cost and the increase of values can often translate into increase of profits for the adopter.

Profit orientation is based on the profitability of OSS. Under this orientation, the adopter is mainly interested in exploiting an OSS by treating the OSS as a source of profits. In another word, the profitability of the OSS for the adopter can only be realized in generating profits from the OSS. The impact of the adoption of OSS on the bottom line of the business is more emphasized in profit orientation than in value orientation. Thus, the adoption of OSS under profit orientation is often associated with analysis of Return on Investment, cash flow, net profit, and other financial measures. Profit orientation leads to an important topic in OSS research: business models based on OSS, as discussed in Section 4.4.

Though the use value of the OSS is often a premise for the profitability of the OSS, profit orientation does not necessarily increase the use value of the OSS. An antithetical example is still Microsoft's notorious purpose of OSS adoption. Obviously, Microsoft

adheres to profit orientation. It strives to gain more profits from the OSS movement by reducing the use value of the OSS. Only by "embracing, extending, and extinguishing" can Microsoft earn the maximum profits from monopoly. However, for most adopters under the profit orientation, to increase the use value of the OSS still dovetail their own best benefits. Hence, "embracing, contributing, and prospering" is a wise strategy for these adopters.

The adoption cases in previous research on IT adoption are all based on value orientation. This is understandable because subjectively previous authors did not recognize the perspective of orientation, and objectively the proprietary characteristics of candidate IT constrained its own Strategic Usage. From this fact, research on IT adoption which considers profit orientation is a void to fill.

5.3.2 Participation

The adopters of a software automatically become members of the user community of the software. The interaction between an adopter and the community is normally scarce for proprietary software. And an adopter of a closed source software usually cannot participate in the development of the software. OSS provides an opportunity to OSS adopters in that the OSS adopters can freely interact with the community by participating in all kinds of activities of the community, even including software design and coding.

From the perspective of participation, the interaction between OSS adopters and OSS communities can be classified into two coarse-grained categories: passive and active. Under passive participation, an adopter does not contribute back to the community. The most possible interaction between a passive adopter and the community is that the adopter looks for help from the community and learns from the community.

Unfortunately, passive participation seems to dominate. One example can be found in Mockus et al. (2000). While Apache is the most popular web server, only a few webmasters contributed back to the Apache community (e.g., reported bugs).

Under active participation, an adopter actively contributes back to the community. The contributions can be in many forms, such as bug reports, feature requests, bug fixes, feature additions, documentation, marketing, socializing, etc. The active participation of adopters can greatly boost the vitality of the OSS project.

This perspective of participation was not recognized by previous research on IT adoption since the interaction between the adopter and the community was not taken into consideration in previous studies. However, in OSS, the power of community cannot be ignored. Thus, this novel perspective is quite pertinent to the research of OSS adoption.

5.4 Strategic Usages of OSS

By the intersection of the two novel perspectives – orientation and participation, four kinds of Strategic Usages of OSS are identified: Consumption, Collaboration, Commensalism, and Mutualism (Table 3).

Table 3: Strategic Usages of OSS / A classification of OSS adopters

		Orientation	
		Value	Profit
Participation	Passive	Consumption / Consumer	Commensalism / Profitor
	Active	Collaboration / Prosumer	Mutualism / Partner

Consumption is a kind of Strategic Usage of OSS that is under value orientation and with passive participation. Under value orientation, Consumption aims to exploit the use

value of OSS by directly using the products and services provided by the OSS projects. The adopted OSS is not treated as a source of profit. With passive participation, Consumption brings no direct contribution back to the OSS community.

Collaboration is a kind of Strategic Usage of OSS that is under value orientation and with active participation. Under value orientation, Collaboration aims to exploit the use value of OSS by directly using the products and services provided by the OSS projects. The adopted OSS is not treated as a source of profit. With active participation, Collaboration brings various contributions back to the OSS community.

Commensalism is a kind of Strategic Usage of OSS that is under profit orientation and with passive participation. Under profit orientation, Commensalism aims to generate profits from the adoption of OSS by building business based on the products and services provided by the OSS projects. The adopted OSS is treated as a source of profit. With passive participation, Commensalism brings no direct contribution back to the OSS community. Commensalism is a biological term. It is a type of symbiosis where two (or more) organisms from different species live in close proximity to one another, in which one member is unaffected by the relationship and the other benefits from it (http: //biotech.icmb.utexas.edu /search /dict-search.phtml ?title= Commensalism). As a metaphor, the adopter benefits from the OSS community while the community does not benefit directly from the adopter.

Mutualism is a kind of Strategic Usage of OSS that is under profit orientation and with active participation. Under profit orientation, Mutualism aims to generate profits from the adoption of OSS by building business based on the products and services provided by the OSS projects. The adopted OSS is treated as a source of profit. With

passive participation, Mutualism brings various contributions back to the OSS community. Mutualism is a biological term. It is a type of symbiosis where two (or more) organisms from different species live in close proximity to one another and rely on one another for nutrients, protection, or other life functions. Both (or all) of the organisms involved benefit from the relationship (http: //biotech.icmb.utexas.edu /search /dict-search.phtml ?title= Mutualism). As a metaphor, both the adopter and the OSS community benefit from each other.

5.5 Classification of OSS Adopters

An adopter of OSS could be a person, a business enterprise in the private sector, an organization in the public sector, or other types of entities. As mentioned above, adopters are traditionally categorized by their timeliness of adoption. Although this classification is suitable for the study of diffusion, it ignores other characteristics of the adopters and their adoptions.

This thesis creatively classifies the OSS adopters by their major Strategic Usage of OSS. Accordingly, the adopters of OSS can be labeled as Consumer, Prosumer, Profitor, and Partner (Table 3). Obviously, this classification can be combined with the traditional classification based on Timeliness of Adoption in order to meticulously classify the adopters into more fine-grained categories. However, this kind of complexity is not necessary for this thesis.

The classification of adopters discussed in this section should not be confused with the roles played by community members as presented in Section 4.1. The adopters of an OSS become members of the community of the associated OSS project. Different adopters play various roles in the community according to the contributions they made to

the community. The relationship between types of adopters and the roles they play will be explained in the following sections where each type of adopters is defined. In general, the determinant of the types of adopters – the Strategic Usage of OSS – also dictates the roles the adopters will play.

5.5.1 Consumer

An adopter is a Consumer of OSS if her major Strategic Usage of OSS is Consumption. Like end-user in the case of proprietary software, a Consumer just wants to use the OSS as it is, with no intention (or capability) of modifying or redistributing the codes. Nor will she report bugs, let along contribute patches. The majority of OSS adopters are Consumers (Nakakoji et al., 2002). Obviously, Consumers only play the "Passive User" role in the OSS community.

There is one special kind of Consumer that can be called Constructor. Like licensee of Non-Disclosure Agreement in the case of proprietary software, a Constructor treats the adopted OSS as a functional component of her bigger software system. For example, Linux, Apache, and Zope can all become functional components of a web-based software system. It is noteworthy that the bigger system should not be for sell. If it is for sell, i.e., for generating profits, then the Constructor becomes a Profitor as will be discussed later.

As long as the adopted OSS provides adequate functionality to the whole system, the Constructor cares less about its internal details. The Constructor generally does not modify the adopted OSS, except for necessary customization. Though the thriving of the adopted OSS is beneficial to the Constructor, she generally does not actively contribute to its evolution. Thus, the Constructor acts more like a normal Consumer, under value

orientation and with passive participation. The identification of the Constructor was not recognized by previous OSS research.

As mentioned before, Consumers are the majority of OSS adopters. It is also the only kind of adopters studied in previous tactic level study of IT adoption. The user base of proprietary software could be roughly counted by the number of copies (or licenses) sold. Because OSS can be freely distributed, it is very hard to count the number of Consumers of an OSS. Moreover, there is no reliable record of activity for Consumers in the public archives of OSS projects since Consumers generally do not participate in the exchange of information within the OSS community. They are just lurkers. The count of downloads of releases is the only known statistic that may reflect to some extent the width of Consumers. Thus, research on OSS Consumers is restricted by the lack of reliable data.

5.5.2 Prosumer

An adopter is a Prosumer of OSS if her major Strategic Usage of OSS is Collaboration. The word "Prosumer" was coined in 1980 as a blend of "producer" and "consumer" by Alvin Toffler in his famous book "The Third Wave" (http: //www.quinion.com /words /turnsofphrase /tp-pro4.htm). Toffler predicted that interconnected users would collaboratively produce products, 10 years before the naissance of the World Wide Web (http: //coforum.de /index.php4?Prosumer).

More than testers in the case of proprietary software, a regular Prosumer not only uses the OSS, but also reports bugs, submits feature requests, posts messages to mailing lists or forums, etc. In a word, the Prosumer contributes to the evolution of the OSS project. In this sense, the Prosumer plays the "Active User" role in the OSS community.

The more capable Prosumers even fix bugs by providing patches, or add new functions and features. In this sense, this kind of Prosumers play the various developer roles (Peripheral Developer, Central Developer, Core Developer, and Project Leader) depending on the quantity and quality of their contributions.

Since a Prosumer actively participates in all kinds of activities in the OSS community, her actions are recorded in various public archives. For example, if she submitted patches, the submissions are recorded in the Patch Tracker or Configuration Management Systems. So, there are archived data for studying the behaviors of Prosumers. However, one possible problem is that the Prosumer can be anonymous, use multiple identities, or use fake name in some cases like in posting messages to a forum.

The traditional adoption models such as the Diffusion of Innovations model and the Technology Acceptance Model may not be suitable for the study of Prosumers. Apparently, their independent variables such as usefulness and ease-of-use cannot explain convincingly why Prosumers participate in the development of a specific OSS project. For instance, according to previous research, low ease-of-use may lead to rejection of an IT. In reality, a Prosumer may contribute to an OSS project just because the products produced in the project are not easy to use. As mentioned before, the motivation of OSS developers is in its own a hot topic in OSS research.

5.5.3 Profitor

An adopter is a Profitor of OSS if her major Strategic Usage of OSS is Commensalism. Like acquirer in the case of proprietary software (e.g., Microsoft bought several small software companies which were the initiators of profitable applications), the Profitor treats the adopted OSS as a machine to earn profit. The author of this thesis

coined the word "Profitor" because the existent word "profiteer" has too negative connotations. Someone may argue that profiteer is only applicable to adopters like Microsoft. The occurrences of the word "Profitor" through Google search are only in other languages.

The Profitor may behave like a Constructor in the category of Consumer in that the adopted OSS becomes one component of a bigger software system. However, this bigger system is for sell in the case of a Profitor. Like a Constructor, the Profitor does not involve in the evolution of the adopted OSS. Though not a previously identified model as in Hecker (2002), this Constructor-like Strategic Usage of OSS could be a viable business model. It may be called the "Profitor" model.

Famous examples of this model in practice (DiBona et al., 1999) are proprietary software applications on OSS platforms (e.g., Oracle database on Linux), IBM's adoption of Apache as the web server in its application suite, etc. Here, Linux and Apache are components of proprietary applications (Oracle database uses Linux, not vice versa; IBM's suite uses Apache, not vice versa). Although Oracle and IBM actively participate in the OSS movement in general, it is not required in the Profitor model that the adopter involves in the evolution of the adopted OSS. For many smaller companies that develop proprietary applications that incorporate OSS components such as Linux and Apache, it is more probable that they do not participate in the evolution of Linux or Apache. But they can still earn profits from utilizing Linux or Apache.

Since a Profitor only plays the role of "Passive User" in the OSS community, the symbiosis between a Profitor and the OSS community is Commensalism. It is only the Profitor who benefits directly from the relationship, though the OSS community may

gain indirect benefits such as enlarged user base or marketing effect. Research on Profitors faces the same problem – lack of valid data – as research on Consumers.

The traditional adoption models such as the Diffusion of Innovations model and the Technology Acceptance Model may be suitable for the tactic level study of Profitors. However, further theoretical and empirical evidences are required before any conclusion can be made.

5.5.4 Partner

An adopter is a Partner of OSS if her major Strategic Usage of OSS is Mutualism. Like partners in the case of proprietary software, a Partner actively participates in the evolution of an OSS project for the purpose of earning profits. Many of the previously proposed business models (Hecker 2002) can be utilized by Partners in this Strategic Usage of OSS – Mutualism. For example, the "Support Sellers" model and the "Loss Leader" model are adopted by people like Ousterhout (1999) who are major developers of OSS. They founded commercial companies to provide support, and to sell add-ons for the OSS or more powerful commercial versions.

Since both the Partner and the OSS community benefit from the partnership, the symbiosis between them is Mutualism. A Partner may play multiple roles in the community, even as Core Developer or Project Leader as in the cases like Ousterhout (1999). Like Prosumers, the contributions of Partners are usually recorded in the public archives of OSS projects.

The traditional adoption models such as the Diffusion of Innovations model and the Technology Acceptance Model may not be suitable for the study of Partners. For

instance, the weakness of an OSS in terms of usefulness and ease-of-use may lead to refusal of adoption according to the traditional models. However, this kind of weakness is exactly a valuable opportunity for a Partner who can earn profits by providing more user-friendly version of the OSS. Hence, for a Partner, she may more than happy to adopt this OSS and adapt it to her benefits.

5.6 Implications of Strategic Level Study

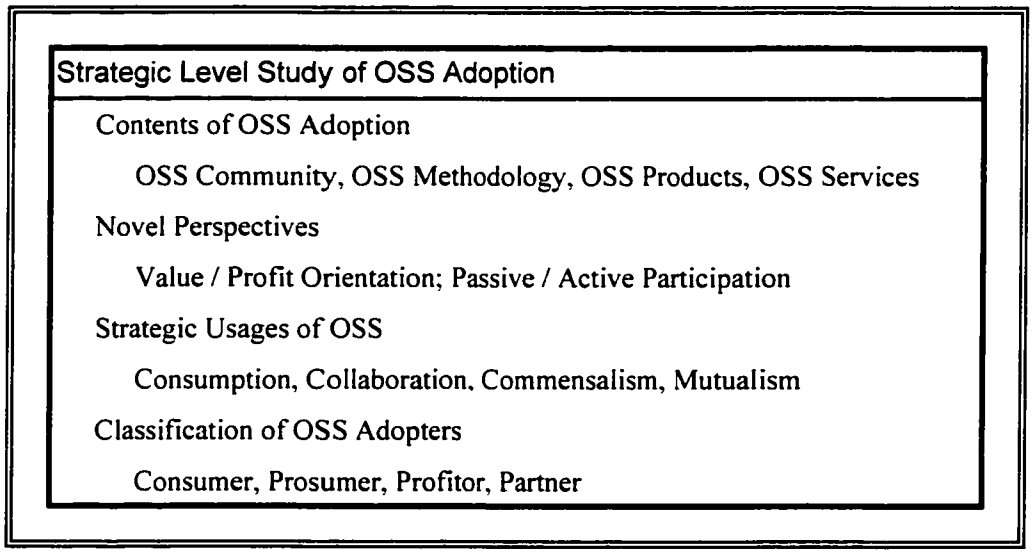
The strategic level study of OSS adoption can be beneficial to both OSS researchers and OSS adopters. As creative as it is, it opens a brand new field for OSS researchers. Through the strategic level study of OSS adoption, a potential adopter can recognize the full contents of OSS adoption, choose viable Strategic Usages of OSS, and identify herself as one or more appropriate types of adopters. Especially, the strategic level study of OSS adoption can behave like Business Process Reengineering for organizational adopters. It may inspire new ideas for business development. Of course, not all Strategic Usages of OSS are suitable for all potential adopters. Constrained by their capability and environment, many potential adopters are inclined to just one Strategic Usage of OSS. It is understandable that most potential individual adopters who are not technically capable will naturally choose Consumption as their mode of OSS adoption.

5.7 Summary

Section 5 first pinpointed certain limitation of two prevailing theoretical models in the research of IT adoption in the context of OSS adoption. It suggested that OSS adoption study should be carried out at two levels: strategic level and tactic level. It then proceeded with the strategic level study of OSS adoption.

Figure 6 shows a conceptual diagram of strategic level study of OSS adoption. The contents of OSS adoption should include all four components of an OSS project, not just software products as in previous IT adoption study. Two new perspectives to OSS adoption study were suggested: the Orientation perspective and the Participation perspective. Four viable Strategic Usages of OSS were identified by the intersection of the two new perspectives: Consumption, Collaboration, Commensalism, and Mutualism. A new classification of adopters was proposed according to their major Strategic Usage of OSS: Consumer, Prosumer, Profitor, and Partner. And the implications of the strategic level study of OSS adoption were briefly described.

Figure 6: A conceptual diagram of strategic level study of OSS adoption



6 Tactic Level Study of OSS Adoption

Though OSS is often compared with proprietary software, competitions among OSS projects are as fierce as in closed source software. One famous example is the fork of the BSD Unix into three OSS projects: FreeBSD, OpenBSD, and NetBSD. When the appropriate Strategic Usage of OSS has been chosen, the potential adopter then needs to conduct tactic level study of OSS adoption by assessing candidates of OSS projects. For example, a potential Prosumer needs to systematically evaluate each candidate before finally chooses the best one to join.

As pointed out before, previous IT adoption research is carried out at the tactic level by analyzing potential adopters' perception of the characteristics (e.g., usefulness, ease of use) of the IT. However, there is not yet an OSS research which uses this traditional method. For the advocates of OSS, the question of whether to adopt OSS is often answered by stating that OSS is generally superior to closed source software. Many defended OSS by qualitative and anecdotal arguments (e. g. Pfaff 1998; Raymond 2001). Wheeler (2002b) did an exceptional job by compiling results from numerous quantitative surveys, experiments, and other studies that compared OSS with proprietary software on market share, reliability, performance, scalability, security, total cost of ownership, and other issues. He proved that OSS was at least as good as or better than its proprietary competition in certain situations.

However, most of these kinds of quantitative evidences are concerned with only a few high profile OSS projects such as Linux and Apache since they are the focus of previous academic and industrial studies on OSS. If a potential adopter wants to evaluate

a regular OSS project which is less recognized, all previous statistics about those high profile projects provide little help. Since there is generally no existing comparison, the potential adopter has to evaluate these low profile projects on a case by case basis. Thus, a project-oriented assessment framework is required as a decision support model for OSS adopters. Adopters can use this framework to systematically evaluate any OSS project, not just those famous ones. This section strives to propose such an assessment framework for OSS projects.

6.1 Project-Based Assessment Framework

Unfortunately, there is a scarcity of research on assessment framework for OSS projects. The existing framework proposed in Wang and Wang (2001) abide by the traditional product orientation. By mainly evaluating the software products per se, they largely ignore the rich context of the products: OSS projects and OSS communities. Nonetheless, they have their merits.

Wang and Wang (2001) adopted an insightful approach to discuss OSS adoption. They considered the fact that most criteria for choosing a suitable software package have been extensively studied and should be applied no matter the candidate is open source or closed source. Thus, they did not cover such important adoption criteria, for example, functional capability, efficiency, speed of execution, and organizational standards and preferences. Instead, they proposed a product-oriented evaluation framework to compare and analyze distinctive features of OSS candidates.

They discussed 5 issues in technical requirements (availability of technical support, future functional upgradability, open-standard compatibility, customizability and extensibility, and high reliability) and 4 issues in management requirements (budgetary,

development team expertise, licensing and project scope, and long-term maintainability). Then, they listed 10 criteria against which 20 OSS products were assessed. The technical criteria are: technical support, backward compatibility, standard compatibility, binary availability, integration with commercial software, commercial adoption, and operating system dependency. The managerial criteria are: software license, current development status, and commercial substitutes.

There are still moot points in their requirements and criteria. First, most of these requirements and criteria are not unique to OSS; they should also apply to closed source software. Second, not all of their requirements have corresponding criteria, such as development team expertise (refers to the team of adopter, not the developers of OSS candidates). Third, some requirements and criteria are in the wrong category. For instance, long-term maintainability should be in the same category as future functional upgradability, while commercial adoption (the extent to which the OSS has been commercially adopted) should be a managerial criterion. Fourth, some criteria do not relate to any requirement, such as commercial substitutes. Last but certainly not the least, the utility of some criteria in a decision model for OSS adoption is open to questions.

- The criterion of binary availability is too trivial. On one hand, it would be redundant to ask this question to closed source software since the only available form of delivery is in binary. On the other hand, most popular OSS projects do provide binary packages for most popular platforms. Anyway, for OSS, the binary package can be obtained by compiling the source files. And one advantage of OSS is that users can build up their own version of binary executables by including only pertinent modules. This advantage can be extended to meet one of the technical

requirements suggested by Wang and Wang (2001): customizability and extensibility. Since the source codes are available, users can customize and extend the OSS with such a freedom that is not imaginable for closed source software.

- The criterion of commercial substitutes (whether commercial substitutes exist) is useless. In reality, OSS is often considered as a substitute of commercial software, not vice versa. If there is no substitute, no matter commercial or open source, there is no need for further evaluation since there is no choice. If there is any substitute, no matter commercial or open source, the whole set of criteria should be applied to both the candidate and its all substitutes in order to select the best one. Then the criterion of existence of commercial substitutes is just superfluous.

- The criterion of integration with commercial software should be changed to integration with other software, including both open source and closed source, which could interact with the candidate. From this criterion and that of commercial substitutes, we can see that Wang and Wang did not treat OSS and commercial software equally. They implied that commercial software was the norm and OSS was the exception.

- The criterion of standard compatibility is more pertinent to closed source software than to OSS. Standardization is as important to software industry as laws to society. OSS usually gains high points in this criterion because full compliance to international standards is its basic instinct to survive and thrive. It is impossible for OSS to maintain a proprietary standard since the source codes will reveal the details of that standard to the whole world (Pfaff 1998). In contrast, commercial software giants like Microsoft and Adobe strive to establish their own proprietary standards as

the de facto standards. They have the inclination to lock users to their own territory of technology. There is also a fault in Wang and Wang's operationalization of this criterion. They only list 10 assorted standards and one item for "no standard applicable". Logically, there should be one item for "other standards".

Having pinpointed the weaknesses of the framework of Wang and Wang (2001), this thesis adopts their idea to focus on criteria that are particularly relevant to OSS. Furthermore, this thesis suggests that these criteria should be assessed on the accumulated data in the public archives of OSS projects. A Chinese maxim tells us: facts speak louder than rhetoric. The public archives of OSS projects contain inarguable facts about the history and status quo of the projects. These pertinent facts are valuable in assessing the projects.

Obviously, an OSS project is much richer in contents than the OSS products produced in this project. An assessment framework should be project-oriented, rather than product-oriented as in Wang and Wang (2001). This research proposes an assessment framework for OSS projects. This project-oriented framework carries out systematic evaluation of an OSS project based on historical data, available from public archives of the project. It contains 9 components that are grouped into three categories: Managerial Analysis, Technical Analysis, and Supportive Analysis. Each component analyzes one or more metrics that measure various aspects of the project.

The metrics suggested in this framework are not meant to be exhaustive or indispensable, but to be heuristic. They are just examples of what kinds of analyses can be conducted. Data sources for these analyses are indicated since one characteristic of this assessment framework is the utilization of the public archives of the OSS projects.

Relevant propositions are formulated for future study. Empirical research questions are posed related to the components. These questions will be answered by analyzing empirical data in Section 7 later.

From the strategic level study of OSS adoption, we know that not all adopters are created equal. Each kind of adopters, classified by their Strategic Usage of OSS, may have divergent requirements and objectives. Weighting and interpretation of criteria in the assessment framework certainly depend on the specificity of the adopter. Nuances in criteria for assessing OSS projects among Consumers, Prosumers, Profitors, and Partners should be an interesting research topic. Unfortunately, this topic is a void to fill since adopters are traditionally classified in terms of timeliness of adoption in the literature. This thesis is an initial attempt to pinpoint the implications of different types of adopters to the assessment framework of OSS projects.

6.2 Summative Subjective Metrics

Summative subjective metrics are widely used in the proposed assessment framework. Usually they are in the form of "the degree to which an objective measurement of the project meets the adopter's requirement". For instance, the metric "Conformance of Development Status" tests the degree to which the Development Status of the project meets the adopter's requirement. It is subjective because it is based on the judgment of the adopter. It is summative because the judgment of the adopter is based on a summative evaluation of an objective measurement.

There are two reasons for the wide use of summative subjective metrics. First, this thesis emphasizes that different kinds of adopters have divergent requirements. Using summative subjective metrics can accommodate these divergent requirements under a

simplified uniform. For example, a Consumer may prefer a project which is at either stable or mature stage while a Prosumer may prefer a project which is at an earlier stage such as alpha. It is possible to use an objective metric "Development Status of the Project" with category values ranging from planning to mature. But then the same value may have different denotations for different adopters. For a Consumer, the value "alpha" may receive an evaluation score of 50 while the value "mature" with an evaluation score of 100. For a Prosumer, results would be the opposite. The summative subjective metric "Conformance of Development Status" elegantly accommodates these two situations and other possible situations for the Profitor and the Partner in a simple and consistent form. The values of this summative subjective metric have the same denotation for all types of adopters. For example, a 10 from this metric always mean that the development status fully meets the requirements of adopters, no matter what type of adopters.

Second, in previous adoption studies, likeart points self-reported by subjects were uniformly used as the sole measurements for both independent and dependent variables. There are only a few exceptions for the dependent variables. For instance, Szajna (1994) used the subject's actual choice among several software packages as the dependent variable, instead of using the normal dependent variable "self-reported intention to use a particular software package". In another case (Thong 1999), one dependent variable "likelihood of IS adoption" was measured dichotomously by "if the business is computerized", and another dependent variable "extent of IS adoption" was measured by the number of personal computers and the number of software applications used in the business.

One improvement of the summative subjective metrics proposed in this thesis over the subjective metrics used in previous adoption studies is that the former is based on objective measurements while the latter is solely subjective. For instance, the metric "Conformance of Development Status" is based on an objective measurement of the project – the Development Status. On the contrary, the metrics "Perceived Usefulness" and "Perceived Ease of Use" used in a plethora of studies based on the Technology Acceptance Model relied only on subjective agreements to statements like "Learning to operate MRP software was easy" and "I find MRP software useful in my work" (Gefen 2000, p. 73). There was no objective measurement of "ease" or "usefulness".

6.3 Managerial Analysis

Managerial Analysis investigates the philosophical, legal, and financial issues that determine the behavioral regulations of the OSS project. It comprises Goal Analysis, License Analysis, and Value Analysis.

6.3.1 Goal Analysis

Goal Analysis is to determine if the objectives and methodology of the OSS project meet the requirements of the adopter. It concerns about what objectives the project strives to fulfill, and the methodology to fulfill the objectives.

Naturally, the first question a potential adopter should ask is "what kinds of applications / products / services does the OSS project plan to provide?" If the target functionality of the application cannot fit the requirements of the adopter, there is no further need to investigate other questions like "how well does the application implement

the functionality?". Thus, the objectives of the project should be the first thing to be assessed.

The methodology of OSS projects is quite different from the traditional methodology for developing proprietary software. As mentioned in Section 4.2, the OSS development model and its comparison with traditional model is a hot topic in OSS research. However, detailed review of this topic would require a separate work. So it is beyond the scope of this thesis. This thesis only points out that there are also nuances and variances in terms of software development methodology / process among OSS projects. For example, Nakakoji et al. (2002) described the differences in developing process among four OSS projects.

Although there is already international standard (ISO / IEC 15504) on Software Process Assessment (Emam & Jung, 2001), this standard is based on the traditional closed source software development model. Thus, research on software process assessment for OSS development model is a void to fill. Nevertheless, the assessment of methodology should be carried out by the adopters at least subjectively.

Due to its trait, Goal Analysis is mostly qualitative, specific, and summative. This analysis can be a veto criterion. If an OSS project cannot pass this analysis, it will be discarded.

6.3.1.1 Data Sources

There is no conventional place in the public archives for the descriptions of project objectives, philosophy, and methodology. They may be scattered among project web site, design documents, messages in mailing lists and forums, and other archives.

6.3.1.2 Implications for Adopters

Compared to objectives, methodology of the project may not be salient for Consumers and Profitors since they do not participate too much in the OSS community. However, for a potential Prosumer, the methodology of the OSS project can make differences. As a volunteer who can choose which project to join, a Prosumer will certainly choose a project with agreeable methodology. Similarly, for a potential Partner, the methodology of the project must meet the taste of the Partner.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 1: When carrying out Goal Analysis of OSS projects, different types of users may evaluate differently an identical case of goal.

Proposition 2: When carrying out Goal Analysis of OSS projects, Prosumers and Partners will assign higher weights to the assessment of the methodology than will Consumers and Profitors.

6.3.1.3 Metrics

This thesis suggests two summative subjective metrics for the Goal Analysis of an OSS project. They are Conformance of Objectives, and Conformance of Methodology.

6.3.1.3.1 Conformance of Objectives

Conformance of Objectives compares the objectives of the OSS project to the managerial requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the objectives of the OSS barely meet the requirements of the

adopter, while 10 means the objectives of the OSS fully meet the requirements of the adopter.

6.3.1.3.2 Conformance of Methodology

Conformance of Methodology compares the methodology of the OSS project to the philosophical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the methodology of the OSS barely meets the requirements of the adopter, while 10 means the methodology of the OSS fully meets the requirements of the adopter.

6.3.1.4 Empirical Research Question

Some authors deemed that the majority of software developed under OSS development model were system software and programming tools, and the results of developing other kinds of software were marginal (Sandred 2001, p. 145). Others noticed that there were an increasing number of productivity and entertainment applications (Feller & Fitzgerald, 2000). Empirical data can then tell in what kinds of application areas OSS projects already burgeon. Thus, one empirical research question is pertinent:

Q1. What types of applications do OSS projects provide?

6.3.2 License Analysis

The aim of license analysis is to determine if the license(s) adopted by the OSS project can meet the requirements of the potential adopter. As explained in Section 4, licenses are regarded as identification cards of OSS.

Bruce Perens provided a classical analysis of OSS licenses in DiBona et al. (1999). This section is derived from his work. There are 6 major kinds of licenses adopted in

OSS: GPL-like, LGPL-like, MPL-like, BSD-like, dual / multiple licenses, and others. The gist of difference among these licenses is the degree by which derived source codes may be taken proprietary. The first four kinds of licenses can be placed on a spectrum from most to least in terms of restriction on derived codes. GPL-like licenses leave no room for proprietary codes; while BSD-like ones give most liberty. Though Perens mentioned dual / multiple licenses as a license strategy, he did not treat it as a separate category for licensing practice in OSS. Actually, dual / multiple licenses may be a wise choice that caters to the needs of both OSS community and commercial entities. Netscape is an example of prominent companies who placed triple licenses (MPL / GPL /LGPL, or NPL / GPL /LGPL) on the famous Mozilla web browser project (http: //www.mozilla.org /MPL /relicensing-faq.html).

It is noteworthy that License Analysis is much more significant for OSS than for proprietary software. For OSS, License Analysis may be anything but a cost issue. On one hand, it is an ideological issue and a business strategy issue. On the ideological side, different licenses imply different philosophies. For example, GPL reflects the philosophy of the Free Software Foundation (http: //www.gnu.org) so it is also a political manifesto (DiBona et al., 1999). On the strategy side, different licenses impose different restrictions to certain business behaviors. For instance, GPL-like licenses do not allow the derived codes becoming proprietary. The case of KDE, Qt, and Troll Tech described by Bruce Perens (DiBona et al., 1999) is a vivid demonstration of the power of licenses on business strategy. On the other hand, cost of license can be ignored for OSS since there is simply no license fee. For proprietary software, License Analysis is nothing but a cost issue. It simply turns out to be a cost analysis.

Due to its trait, License Analysis is mostly qualitative, specific, and summative. This analysis can be a veto criterion. If an OSS project cannot pass this analysis, it will be discarded.

6.3.2.1 Data Sources

The licenses adopted by the OSS project should be indicated on the project web site. They are also indicated in the source codes. The source codes can be obtained via Configuration Management Systems, Patch Track, and releases.

6.3.2.2 Implications for Adopters

The nuances among different adopters take significant effect in License Analysis. For a Consumer who has no intention to modify or distribute the codes, any license may be ok as long as it allows free usage. For a pure Prosumer who wants her contributions shared by the community rather than exploited by some commercial entities, GPL-like license is the preferred choice as shown in previous studies (e.g., Wheeler 2002a). For a Profitor who normally does not need to modify the codes, any license conforming to the Open Source Definition allows her to aggregate OSS components with the parts developed by herself. For a Partner, BSD-like or dual /multiple licenses can provide much needed flexibility for commercial opportunities.

Wheeler (2002a) gave an inspiring example on license issue. The open source security component Kerberos became a victim of Microsoft's infamous maneuver of "embracing, extending, and extinguishing", because it has adopted a BSD-like license. It is then easy to understand why Microsoft treated the GPL license as its archenemy.

Kerberos could not be exploited by Microsoft if it had been released under a GPL-like license.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 3: When carrying out License Analysis of OSS projects, different types of users may evaluate differently an identical case of licenses.

Proposition 4: When carrying out License Analysis of OSS projects, Prosumers and Partners will assign higher weights to the assessment of the licenses than will Consumers and Profitors.

Proposition 5: When carrying out License Analysis of OSS projects, Consumers and Profitors will less likely have dominant preference for the types of licenses, Prosumers will more likely prefer GPL-like license, and Partners will more likely prefer BSD-like or dual /multiple licenses.

6.3.2.3 Metrics

This thesis suggests one summative subjective metric for the License Analysis of an OSS project. This metric is Conformance of License.

6.3.2.3.1 Conformance of License

Software license is one of the criteria suggested by Wang and Wang (2001). Conformance of License compares the licenses of the OSS project to the legal requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the licenses of the OSS barely meet the requirements of the adopter, while 10 means the licenses of the OSS fully meet the requirements of the adopter.

6.3.2.4 Empirical Research Question

Since License Analysis is significant for OSS projects, empirical data can show what kinds of licenses are actually adopted by OSS projects. Thus, one empirical research question is pertinent:

Q2. What kinds of licenses are used in OSS projects?

6.3.3 Value Analysis

Value Analysis aims to assess the financial feasibility and profitability of the adoption of the OSS project. Value can be regarded as benefit minus cost. Value Analysis then needs to estimate the benefits that would be generated by the Strategic Usage of OSS, and to measure the costs associated with the Strategic Usage of OSS. This analysis can be a veto criterion. If an OSS project cannot pass this analysis, it will be discarded. Since Value Analysis of an OSS project has rich contents that are impossible to cover in one section , this thesis just emphasizes the implication of different adopters to Value Analysis.

6.3.3.1 Data Sources

Value Analysis is the only component in this assessment framework which cannot find enough data from the public archives of OSS projects. On one hand, both benefit and cost of the Strategic Usage of OSS are adopter specific. So most data should be obtained from the adopter side rather than from the OSS project. On the other hand, as discussed in Section 4.2.2, cost related data such as hours spent on the OSS Project are currently not input into the public archives.

6.3.3.2 *Implications for Adopters*

Value Analysis is paramount to all types of adopters. However, different adopters have divergent criteria for benefits and costs, and may require different analysis methods.

For a Consumer, since she does not put efforts back into the OSS community, she is just like a regular user of proprietary software. Thus, popular cost-benefit analysis method such as Total Cost of Ownership (Ferrin & Plank, 2002) could suit her needs. Total Cost of Ownership for an OSS should generally be lower than its proprietary competitors because an OSS has many advantages in cost saving. At least, an OSS has no big price tag attached to its license.

For a Prosumer, since she voluntarily contributes back to the OSS community without financial compensation, her criteria for benefits and costs are usually not just monetary. Accordingly, the method of Value Analysis of a Prosumer should be quite different from those used by other kinds of adopters. The study on motivation of OSS developers may help to explain the value system adopted by a Prosumer.

For a Profitor, she may use the same analysis method as a Consumer. But the idea of value chain may be more useful for considering benefits generated from the Strategic Usage of OSS, since the OSS would be deployed as a component in the whole value chain of the Profitor.

For a Partner, the Strategic Usage of OSS directly affects the bottom line of her business. So the selection of an appropriate business model based on OSS, as discussed in Section 4.4, is an indispensable element of her Value Analysis.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 6: When carrying out Value Analysis of OSS projects, different types of users may prefer different types of Value Analysis techniques.

Proposition 6a: When carrying out Value Analysis of OSS projects, Consumers will more likely use the analysis of the Total Cost of Ownership than will other types of users.

Proposition 6b: When carrying out Value Analysis of OSS projects, Prosumers will more likely use non-monetary cost-benefit analysis than will other types of users.

Proposition 6c: When carrying out Value Analysis of OSS projects, Profitors will more likely use the analysis of Value Chain than will other types of users.

Proposition 6d: When carrying out Value Analysis of OSS projects, Partners will more likely use the analysis of appropriate business models based on OSS than will other types of users.

Proposition 7: Compared with their proprietary competitors, OSS products will generally have lower Total Cost of Ownership.

6.3.3.3 Metrics

This thesis suggests one summative subjective metric for the Value Analysis of an OSS project. This metric is Conformance of Value.

6.3.3.3.1 Conformance of Value

Conformance of Value compares the values that would be generated from the Strategic Usage of OSS to the financial requirements of the adopter. Its data type is

Likeart with possible range from 1 to 10. Value 1 means the values of the OSS barely meet the requirements of the adopter, while 10 means the values of the OSS fully meet the requirements of the adopter.

6.3.3.4 Empirical Research Question

Since Value Analysis is project-specific and adopter-specific, it is hard to be conducted over a large number of OSS projects in the timeframe of this thesis. Thus, Value Analysis will be a topic for future empirical research.

6.4 Technical Analysis

Technical Analysis scrutinizes the technical facets of the products produced by the OSS project. It comprises Execution Analysis, Source Code Analysis, and Evolution Analysis. It should be highlighted that for proprietary software, only Execution Analysis is viable since it only requires binary executables. The availability of source codes is a prerequisite for Source Code Analysis and Evolution Analysis.

6.4.1 Execution Analysis

The utility of a software can only be realized in its execution. Test-driving a software is the most direct way and possibly the most convincing way to evaluate it. Execution Analysis is to assess the functionality, performance, usability, and other quality features of OSS products by executing them. Since it is carried out from the viewpoint of adopters, it can be considered as analysis on the external software quality (Anderson et al., 2002). Many software vendors recognize the importance of execution analysis so they provide trial version for potential users to test their software.

OSS projects facilitate the execution analysis of their OSS products because there are no restrictions like limited time or limited functionality, as normally associated with the trial version of close source software. Potential adopters of OSS projects have the full freedom to test the OSS products at their own wish and paces.

6.4.1.1 Data Sources

Many OSS projects provide compiled binary versions of their products for popular platforms such as Windows, Linux / Unix, Mac OS, etc. If binary versions are not available, the potential adopter can always download source codes either from the Configuration Management Systems or the releases. Then the source codes can be compiled and executed.

6.4.1.2 Implications for Adopters

Execution Analysis is important to all adopters. However, each kind of adopters may have its own emphases and may interpret the results of Execution Analysis differently. For example, user-friendliness could be more important to Consumers than to other kinds of adopters. Any weakness of software revealed in Execution Analysis means negative to Consumers and Profitors. However, the same weakness may mean positive to Prosumers and Partners. Prosumers can contribute to the project by improving this weakness. And Partners may treat this weakness as an opportunity to generate profit by adding value to the project.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 8: When carrying out Execution Analysis of OSS projects, different types of users may evaluate differently an identical case of software execution under the same software and hardware configuration.

Proposition 9: When carrying out Execution Analysis of OSS projects, different types of users may prefer different types of Execution Analysis techniques.

Proposition 10: When carrying out Execution Analysis of OSS projects, Prosumers and Partners will more likely assign favorable scores to revealed weaknesses than will Consumers and Profitors.

As indicated by Wang and Wang (2001), most criteria for choosing a suitable software package have been extensively studied. This thesis does not need to reinvent the wheel. Instead, this thesis emphasizes that different adopters may have different requirements for Execution Analysis and may interpret the same metric value differently. Similar point of view can be found in previous research on external software quality (Anderson et al., 2002), where a general consensus is that factors for external software quality can be decomposed differently depending on the set of users and the intended use of the software.

The adopters can use any previously proposed assessment metrics of software which are abundant in the literature (e.g., Anderson et al., 2002; Leung & Leung, 2002). However, in the end, the adopters have to make subjective judgment to interpret these measurements according to their specific quality requirements. Thus, this thesis suggests only summative subjective metrics for execution analysis.

6.4.1.3 Metrics

This thesis suggests three summative subjective metrics for the Execution Analysis of an OSS project. These metrics are Conformance of Functionality, Conformance of Performance, and Conformance of Usability. Conformance of Functionality concerns what the software can fulfill. Conformance of Performance concerns how well the software furnishes the functionality. Conformance of Usability concerns how easy the user operates and maintains the software.

This kind of choices of summative quality metrics is supported by the literature. After examining previous work on external software quality, Anderson et al. (2002) summarized four factors for external software quality: features, learnability, reliability, and response time. The metrics proposed in this section have advantages over these four factors. Functionality is a more precise name for features. Though Anderson et al. (2002) found that reliability became the primary factor in IS managers' decision model, the concept of performance is broader since it includes reliability, response time, and other attributes like robustness. Similarly, usability is broader than learnability by incorporating other attributes like user-friendliness and maintainability.

6.4.1.3.1 Conformance of Functionality

Conformance of Functionality compares the functionality of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the functionality of the OSS barely meets the requirements of the adopter, while 10 means the functionality of the OSS fully meets the requirements of the adopter. High value in Conformance of Functionality is usually a good sign of the OSS project.

6.4.1.3.2 Conformance of Performance

Conformance of Performance compares the performance of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the performance of the OSS barely meets the requirements of the adopter, while 10 means the performance of the OSS fully meets the requirements of the adopter. High value in Conformance of Performance is usually a good sign of the OSS project.

6.4.1.3.3 Conformance of Usability

Conformance of Usability compares the usability of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the usability of the OSS barely meets the requirements of the adopter, while 10 means the usability of the OSS fully meets the requirements of the adopter. High value in Conformance of Usability is usually a good sign of the OSS project.

6.4.1.4 Empirical Research Question

It is impossible to carry out Execution Analysis across a large number of projects in the time frame of this thesis. So empirical Execution Analysis will be one subject of future research.

6.4.2 Source Code Analysis

Source Code Analysis is an advantage bestowed by OSS whose source codes are accessible to anyone interested. On the contrary, Source Code Analysis is not feasible for closed source software whose source codes are kept as top secret away from public access. Compared to Execution Analysis which examines the external software quality, Source Code Analysis investigates the internal software quality, which is from the point of view of programmers (Anderson et al., 2002). However, users of OSS software can

also use Source Code Analysis to assess the internal software quality, in the shoes of programmers. This is impossible for users of closed source software who can only deploy Execution Analysis to study the external software quality.

In this thesis, Source Code Analysis is complementary to Evolution Analysis. The former concentrates on the status quo of software. So only latest versions of software are scrutinized. The latter emphasizes the history of software. All data from the lifespan of a project may become the subject of study. Though Source Code Analysis is an active research area (Harman et al., 2002), previous studies have not explored frequently the public archives of OSS projects. Fortunately, there are still some pilot studies that demonstrated the value of the public archives. For example, Stamelos et al. (2002) carried out a sort of Source Code Analysis based on current releases of OSS, and Antoniol et al. (2002) conducted a kind of Evolution Analysis based on data retrieved from the lifespan of a project.

Since source codes and evolution history of OSS projects are available to anyone interested, the utility of Source Code Analysis and Evolution Analysis to OSS researchers and OSS adopters is only limited to their diligence. There are many kinds of analyses can be applied to data from the public archives, though not all analyses are necessary for different types of adopters as discussed below.

A good recent example of Source Code Analysis is Stamelos et al. (2002). The authors evaluated 100 C programs against four criteria – testability, simplicity, readability, and self-descriptiveness. Each criterion was related to a specific subset of 10 quality metrics: number of statements, cyclomatic complexity, maximum levels, number

of paths, unconditional jumps, comment frequency, vocabulary frequency, program length, average size, and number of inputs / outputs.

Source Code Analysis and Evolution Analysis have close relation with issues discussed in Section 4.3. For example, when calculating the complexity of programs, the impacts of programming languages should be considered. And documentary structure is a fecund topic for Source Code Analysis and Evolution Analysis.

6.4.2.1 Data Sources

It is the most distinguishing idiosyncrasy of OSS that source codes must be accessible to the public. Source codes can be retrieved from Configuration Management Systems or releases.

6.4.2.2 Implications for Adopters

The importance of Source Code Analysis differs for different adopters. It is paramount for a Prosumer or a Partner to carry out detailed Source Code Analysis, while for a Profitor a high level analysis could be sufficient. And a Consumer may not care about Source Code Analysis at all. The issue of programming language can be a good example. A Consumer usually does not care about the programming languages used in an OSS project. For a skillful Prosumer who wants to play a kind of developer role in the project, programming languages are like her natural languages. She cannot join an OSS project which uses a programming language that she is not familiar with, unless she treats this as an opportunity to learn this language. For a Profitor who has no intention to modify the codes, she may only check if the programming languages of the adopted OSS allow easy integration with her other parts. For a Partner, she must consider whether she

possesses expertise in the programming languages. Otherwise she may not be able to derive profit from the project.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 11: When carrying out Source Code Analysis of OSS projects, different types of users may prefer different types of Source Code Analysis techniques.

Proposition 12: When carrying out Source Code Analysis of OSS projects, Prosumers will assign more weights to Source Code Analysis than will Partners, Partners will assign more weights to Source Code Analysis than will Profitors, and Profitors will assign more weights to Source Code Analysis than will Consumers.

6.4.2.3 Metrics

This thesis suggests one summative subjective metric for Source Code Analysis of an OSS project. This metric is Conformance of Programming Languages. Since requirements of Source Code Analysis vary for different adopters, each adopter should add appropriate metrics for Source Code Analysis accordingly.

6.4.2.3.1 Conformance of Programming Languages

Conformance of Programming Languages compares the programming languages of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the programming languages of the OSS barely meet the requirements of the adopter, while 10 means the programming languages of the OSS fully meet the requirements of the adopter. High value in Conformance of Programming Languages is usually a good sign of the OSS project.

6.4.2.4 Empirical Research Question

As described in Section 4.3.1, programming language is an important feature of OSS projects but it has not been thoroughly studied in previous research. Though Python is chosen as the target programming language in this thesis, it would be useful to know what else programming languages are used in OSS projects, and specifically, what kinds of programming languages are used together with Python in Python projects. Thus, one empirical research question is pertinent:

Q3. What kinds of programming languages are used in OSS projects?

6.4.3 Evolution Analysis

Another benefit bestowed by OSS, Evolution Analysis aims to investigate the evolution of the OSS project. The rich historical data kept in public archives make Evolution Analysis a fruitful area for research. Most previous studies on software evolution focused on the evolution of the products per se (Nakakoji et al., 2002). For example, Godfrey & Tu (2000) studied the growth of Linux kernel. Nakakoji et al. (2002) suggested a broader perspective to examine not only the evolution of OSS products, but also the evolution of the associated OSS communities as well as the relationship between these two types of evolutions. In line with most previous studies, Evolution Analysis in this framework still focuses on the evolution of the OSS products. Incorporating the suggestion of Nakakoji et al., Community Analysis is also an important component of this framework, which will be presented in Section 6.5.1.

Evolution Analysis has close relation with the issues discussed in Section 4.2.1. One obvious task in Evolution Analysis is to examine the development status of the project. It

would be appealing for OSS researcher to check how long it takes for an OSS project to reach each stage, especially the stable stage. This kind of real historical data can give an empirical indication of the development speed of OSS projects. Then the development speed of OSS projects can be compared to similar closed source software projects in order to verify if OSS development model can help heal the chronic disease of the software crisis in development speed.

Another task in Evolution Analysis is to examine the history of releases of the project in order to test the feature of "Release early, release often". Frequent releases could also reflect the development speed of the project. Other kinds of analyses such as those carried out in previous studies (e.g., Antoniol et al., 2002; Godfrey & Tu, 2000) can also be conducted in Evolution Analysis. One kind of analysis that could be especially useful for Prosumers and Partners is the estimate of effort as demonstrated in Koch and Schneider (2002).

6.4.3.1 Data Sources

The history of the evolution of an OSS project is recorded in the public archives of the project. Especially, the history of the evolution of source codes can be restored from Configuration Management Systems or releases.

6.4.3.2 Implications for Adopters

There are nuances among different adopters for Evolution Analysis. Taking the example of development status, a Consumer or a Profitor may prefer the project being at either stable or mature stage. So the products of the projects can be used relatively risk-free. In another word, a Consumer or a Profitor may not want to adopt an OSS project at

riskier early stages such as alpha or beta stage. Contrariwise, a Prosumer may prefer to join a project at an early stage so that she can exert more influence. For a Partner, her preference of the development status of an OSS project depends on potential commercial opportunities associated with different stages.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 13: When carrying out Evolution Analysis of OSS projects, different types of users may prefer different types of Evolution Analysis techniques.

Proposition 14: When carrying out Evolution Analysis of OSS projects, Consumers and Profitors will more likely prefer the projects being at either stable or mature stage than will Prosumers and Partners.

Proposition 15: When carrying out Evolution Analysis of OSS projects, Prosumers will more likely prefer the projects being at an early stage than will Consumers and Profitors.

Proposition 16: When carrying out Evolution Analysis of OSS projects, Partners' preference of the development status of OSS projects will more likely depend on potential commercial opportunities associated with different stages than will other types of users.

6.4.3.3 Metrics

This thesis suggests one objective metrics and two summative subjective metric for the Evolution Analysis of an OSS project. The summative subjective metrics are

Conformance of Development Status and Conformance of Evolution Speed. The objective metrics is Frequency of Stable Releases.

6.4.3.3.1 Conformance of Development Status

Current development status is one of the criteria suggested by Wang and Wang (2001). Conformance of Development Status compares the development status of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1 to 10. Value 1 means the development status of the OSS barely meets the requirements of the adopter, while 10 means the development status of the OSS fully meets the requirements of the adopter. High value in Conformance of Development Status is usually a good sign of the OSS project.

Adopters may have their own definition of Development Status. However, the classification used at SourceForge.net can be a good reference, which specifies seven stages: Planning, Pre-Alpha, Alpha, Beta, Stable, Mature, and Inactive.

6.4.3.3.2 Frequency of Stable Releases

As discussed before, frequent releases are a landmark of OSS projects. Frequency of Stable Releases is aimed to quantitatively measure this trait. Its operationalization is the number of stable releases divided by the project's lifespan in terms of months. Its data type is of course real. A high value of Frequency of Stable Releases generally indicates the high vitality and productivity of the project.

6.4.3.3.3 Conformance of Evolution Speed

Conformance of Evolution Speed compares the evolution speed of the OSS to the technical requirements of the adopter. Its data type is Likeart with possible range from 1

to 10. Value 1 means the evolution speed of the OSS barely meets the requirements of the adopter, while 10 means the evolution speed of the OSS fully meets the requirements of the adopter. High value in Conformance of Evolution Speed is usually a good sign of the OSS project.

Adopters may have their own measurements of development speed. However, the Frequency of Stable Releases is usually a good measurement of development speed. But different users may interpreter the same Frequency of Stable Releases differently. Take the example of one stable release per two months, Consumers and Profitors may deem this high frequency an annoyance since they have to upgrade and update too often. So they may give a low score to the Conformance of Evolution Speed. But Prosumers and Partners may deem this high frequency a good sign of productivity. So they may give a high score to the Conformance of Evolution Speed.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 17: When carrying out Evolution Analysis of OSS projects, different types of users may assign different scores of Conformance of Evolution Speed to an identical case of Frequency of Stable Releases.

6.4.3.4 Empirical Research Question

The knowledge gained from a study of a large sample of OSS projects can help the assessment of any specific OSS project. Empirical data can show the characteristics of the evolution of OSS projects. Thus, one research question is pertinent:

Q4. What are the characteristics of the evolution of OSS projects?

6.5 Supportive Analysis

Supportive Analysis examines the vitality of the OSS community and the support and help that can be obtained from the OSS community. It comprises Community Analysis, Documentation Analysis, and Tool Usage Analysis.

The OSS community to an OSS project is like the ground to the giant Antaeus in the Greek and Roman mythology. The OSS community is the source of strength for an OSS project. Similarly, OSS researchers and adopters can also gain strength from the OSS community, just like Antaeus touched the ground. The OSS community is where the researchers and adopters can obtain support and help. From their experience, Davis et al. (2000) became distrustful of commercial support provided by commercial software vendors in general. Instead, they appraised the support provided by open source community. Documentation is one important source of support. And the exchange of information among the OSS community is carried out via various tools.

6.5.1 Community Analysis

The aim of Community Analysis is to assess the vitality of the OSS community by studying the behaviors of the members of the community. It is an analysis of the human resources of the OSS project. As stated in Section 4.1, a member of an OSS community can play various roles depending on her contributions. Because members voluntarily join or leave the community and they voluntarily make different contributions, the OSS community is always in dynamic change. This dynamics makes research on OSS community more challenging than research on a development team of a closed source software in a software company. The latter is relatively more constant in terms of membership and roles.

6.5.1.1 Data Source

As mentioned before, it is hard to obtain reliable data from the public archives on those members of an OSS community who play the role of Passive User. Other methods of data collection, such as web survey, may be deployed to gather data directly from Passive Users. For members who play other roles, their participation is normally recorded in various public archives of the project. Thus, the public archives are a valid data source for the activities of those members.

6.5.1.2 Implications for Adopters

Community Analysis is important to all kinds of adopters. Consumers and Profitors need to obtain support from the community. So the vitality of the community is vital to them. A Prosumer may use Community Analysis differently. She may choose different roles to play according to the demand and supply in the community and her own capability and capacity. A Partner should investigate the community carefully since her profits depend on the cooperation of the community. Like pinpointed before in other issues, a weakness of the community can become an opportunity for a Partner. If the Partner finds a promising project but the community seems to lack certain kinds of human resources such as technical writers for documentation, the Partner can put in her own resources and try to generate profits from the added value produced by her resources. Since different types of adopters may see and interpret the same situation of an OSS community differently, both objective and subjective metrics are used in Community Analysis.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 18: When carrying out Community Analysis of OSS projects, Prosumers and Partners will more likely assign favorable scores to revealed weaknesses than will Consumers and Profitors.

6.5.1.3 Metrics

This thesis suggests four objective metrics and one summative subjective metric for the Community Analysis of an OSS project. The summative subjective metric is the Vitality of Community. The objective metrics are the Number of Active Users, Number of Peripheral Developers, Number of Central Developers, and Number of Core Developers.

6.5.1.3.1 Number of Active Users

Number of Active Users is aimed to measure the quantity of members who play the role of Active User in the project. Its operationalization is to count the number of Active Users for the whole lifespan of the project. Its data type is of course integer. High value in Number of Active Users is usually a good sign of the vitality of community.

As mentioned before, users may use fake names or be anonymous. So it could be difficult to accurately count the exact number of Active Users. This problem can also incur to the following similar metrics.

6.5.1.3.2 Number of Peripheral Developers

Number of Peripheral Developers is aimed to measure the quantity of members who play the role of Peripheral Developers in the project. Its operationalization is to count the number of Peripheral Developers for the whole lifespan of the project. Its data type is of

course integer. High value in Number of Peripheral Developers is usually a good sign of the vitality of community.

Since the classification of various developer roles in OSS community is according to the frequency of contribution, the standard to determine what kinds of frequency can be considered as irregular for Peripheral Developers, regular for Central Developers, and extensive for Core Developers and Project Leaders should be project specific.

6.5.1.3.3 Number of Central Developers

Number of Central Developers is aimed to measure the quantity of members who play the role of Central Developers in the project. Its operationalization is to count the number of Central Developers for the whole lifespan of the project. Its data type is of course integer. High value in Number of Central Developers is usually a good sign of the vitality of community.

6.5.1.3.4 Number of Core Developers

Number of Core Developers is aimed to measure the quantity of members who play the role of Core Developers in the project. Its operationalization is to count the number of Core Developers for the whole lifespan of the project. Its data type is of course integer. High value in Number of Core Developers is usually a good sign of the vitality of community.

There is no metrics for Passive Users and Project leaders since the former have no transaction record in the public archives, and the latter are usually just a few elites of the Core Developers (often just the originator of the project).

6.5.1.3.5 Vitality of Community

Vitality of Community is a comprehensive perception of the vitality of the OSS community by potential adopters. So it is summative and subjective. Its data type is Likeart with possible range from 1 to 10. Value 1 means the OSS community has little vitality, while 10 means the OSS community has sufficient vitality. High value in Vitality of Community is usually a good sign of the OSS project.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 19: When carrying out Community Analysis of OSS projects, different types of users may assign different scores of Vitality of Community to an identical case of community.

6.5.1.4 Empirical Research Question

Development team is the most critical success factor of an OSS project. One of the advantages of OSS development model is that anyone interested can contribute to the project. The benefits of OSS development model may be better realized in projects with a big team, for which management overhead could be significant if under traditional development model (Feller & Fitzgerald, 2000). One example of such big project is the FreeBSD project. There are approximately 200 developers in the FreeBSD project who have commit authority, while 1200 external contributors whose patches are added via these committers (Jorgensen 2001).

However, all open source projects compete for limited resources – talented programmers (Sandred 2001, p. 155). Not every project can attract a big team of

developers. Actually, Krishnamurthy (2002) already found that one-person team was not uncommon.

Empirical data can then tell the norm and mode of development teams in OSS projects. Thus, one research question is pertinent:

Q5. What are the characteristics of development teams in OSS projects?

6.5.2 Documentation Analysis

Documentation is one of the most important sources of support to users. New developers can also learn a lot from good quality design documents. It was argued that OSS developers were more interested in high profile activities such as adding new functions, and less interested in low profile activities such as documentation (Raymond 2001). As a result, documentation often lags behind development and is inadequate. For example, there is often a lack of introductory or tutorial material (Ousterhout 1999). Documentation Analysis is aimed to assess the quality and quantity of design documents and user documents in an OSS project.

6.5.2.1 Data Source

As stated in Section 3.7, there is not a standard archive for documentation. However, the web site of the project usually provides links to project documentation.

6.5.2.2 Implications for Adopters

Documentation is important to most adopters. However, different adopters have divergent requirements for certain documents. For example, a Consumer is concerned more about user manual and tutorial; while a Prosumer is concerned more about design documents. Thus, both objective and subjective metrics are used.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 20: When carrying out Documentation Analysis of OSS projects, different types of users may evaluate differently an identical case of documentation.

Proposition 21: When carrying out Documentation Analysis of OSS projects, different types of users may pay special attention to different types of documentation.

6.5.2.3 Metrics

This thesis suggests two objective metrics and one summative subjective metric for the Documentation Analysis of an OSS project. The two objective metrics are Diversity of Documentation and Richness of Documentation. The summative subjective metric is Sufficiency of documentation.

6.5.2.3.1 Diversity of Documentation

Diversity of Documentation is aimed to measure the breadth of project documentation. Its operationalization is to count the number of types of documents. Its data type is of course integer. The classification of documents is adopter-specific since different adopters have different requirements for specific documents. This thesis suggests a coarse-grained classification as follows:

- Marketing Documents: documents for marketing purpose. This kind of documentation was not explicitly recognized by previous OSS research.
- Design Documents: technical documents for developers.
- User Documents: user manual, tutorial, etc.

High value in Diversity of Documentation is usually a good sign of the project documentation.

6.5.2.3.2 Richness of Documentation

Richness of Documentation is aimed to measure the quantity of project documentation. Its operationalization is to count the number of documents. Its data type is of course integer. High value in Richness of Documentation is usually a good sign of the project documentation.

6.5.2.3.3 Sufficiency of Documentation

Sufficiency of Documentation is a comprehensive perception of project documentation by potential adopters. So it is summative and subjective. Its data type is Likeart with possible range from 1 to 10. Value 1 means the OSS project has no pertinent documentation, while 10 means the OSS project has sufficient pertinent documentation. High value in Sufficiency of Documentation is usually a good sign of the project documentation.

6.5.2.4 Empirical Research Question

Certainly, the actual data gathered from public archives of OSS projects can provide statistics for the two objective metrics: Diversity of Documentation and Richness of Documentation. Moreover, this thesis suggests a possible fertile area to be investigated: the relationship between documentation and community. This area has not been explicitly studied in previous OSS research. A virtuous circle could be established between project community and project documentation. A strong community, especially in terms of developers and other contributors like technical writers, could produce more useful

documents. And the richness of documentation may attract more members to join the community. This hypothesis of virtuous circle could be tested using empirical data.

As stated before, there is not a standard archive for project documentation. So collecting data on documentation across a large sample of OSS projects could be difficult. As it will be mentioned in Section 7, it is a pity that the data file provided by SourceForge.net does not contain any information on documentation. So no empirical evidence is obtained for Documentation Analysis in this thesis. The hypothesis of virtuous circle between community and documentation can only be tested in future research.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 22: For OSS projects, Vitality of Community is positively related to Sufficiency of Documentation.

6.5.3 Tool Usage Analysis

Tool Usage Analysis investigates the usage of various tools in the OSS project. Tools are as important as the methodology. Without appropriate tools, the methodology can hardly be applied in practice. In stark contrast to the well-known failure of expensive CASE tools, OSS software engineering tools are widely applied in OSS projects because they fit the characteristics of OSS and OSS development model (Fogel & Bar, 2002; Robbins 2002). Moreover, OSS tools help define the OSS methodology because the open source culture and methodology are demonstrated in the usage of these tools on existing projects (Robbins 2002).

The common tools used in OSS projects provide services like source code management, issue tracking, discussion, design and code generation, Integrated Development Environment, automated testing, and packing and deployment (Robbins 2002). One feature of many tools is that they keep archives of all historical information. There is a one-to-one relationship between these tools and their public archives as described in Section 3.

Tool usage is a good indicator of the vitality of the OSS community. Advanced analysis can be carried out across several tool archives. For example, Mockus et al (2000) studied defect density in the Apache project by collecting data from mailing lists dedicated to CVS and bug reports.

No comprehensive quantitative analysis has been done in previous studies for the usage of a wide range of OSS tools across a wide range of OSS projects. This thesis will provide such needed summary and analysis in Section 7.

6.5.3.1 Data Sources

As stated before, the tools used in OSS projects usually keep archives of different kinds of transactions. The usage of tools can be measured based on data from the corresponding public archives.

6.5.3.2 Implications for Adopters

Tool Usage Analysis is useful to most adopters. However, different adopters may pay special attention to divergent tools. A Consumer may be concerned more about the Support Tracker and mailing lists / forums dedicated to support which are the major source of technical support. A Prosumer may watch carefully the Bug Tracker, the

Feature Request Tracker, and mailing lists / forums dedicated to developers. A Profitor may check frequently the Bug Tracker. A Partner may need to monitor all tools to follow the evolution of the project. Thus, both objective and subjective metrics are used.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 23: When carrying out Tool Usage Analysis of OSS projects, different types of users may evaluate differently an identical case of tool usage.

Proposition 24: When carrying out Tool Usage Analysis of OSS projects, different types of users may pay special attention to different tool usages.

6.5.3.3 Metrics

This thesis suggests two objective metrics and one summative subjective metric for the Tool Usage Analysis of an OSS project. The two objective metrics are Diversity of Tool Usage and Volume of Tool Usage. The summative subjective metric is Sufficiency of Tool Usage.

6.5.3.3.1 Diversity of Tool Usage

Diversity of Tool Usage is aimed to measure the breadth of tool usage. Its operationalization is to count the number of types of tools used. Its data type is of course integer. The classification of tools is adopter-specific since different adopters pay special attention to different tools. A reference classification is introduced in Section 3 since the tools are associated with the public archives. Moreover, adopters should further divide the tool categories into contents-oriented sub-categories such as support and

development. High value in Diversity of Tool Usage is usually a good sign of the vitality of the community.

6.5.3.3.2 Volume of Tool Usage

Volume of Tool Usage is aimed to measure the intensity of information exchanges via the tools. Its operationalization is the average monthly transactions for adopter-interested tools. For tools like CVS, the transactions are commits. For mailing lists or forums, the transactions are messages posted. Its data type is of course real. High value in volume of Tool Usage is usually a good sign of the vitality of the community.

6.5.3.3.3 Sufficiency of Tool Usage

Sufficiency of Tool Usage is a comprehensive perception of tool usage by potential adopters. So it is summative and subjective. Its data type is Likeart with possible range from 1 to 10. Value 1 means the OSS project has very little tool usage, while 10 means the OSS project has sufficient tool usage. High value in Sufficiency of Tool Usage is usually a good sign of the vitality of the community.

6.5.3.4 Empirical Research Question

Empirical data can show the real usage of tools by the OSS community. Thus, one research question is pertinent:

Q6. What are the usages of various tools in OSS projects?

Similar to the discussion in Section 6.5.2.4, this thesis suggests a possible fertile area to be investigated: the relationship between tool usage and community. A virtuous circle could be established between project community and project tool usage. A strong community, especially in terms of developers and other contributors, could produce

higher tool usage. And the highly productive tool usage may attract more members to join the community. This hypothesis of virtuous circle could be tested using empirical data.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 25: For OSS projects, Vitality of Community is positively related to Sufficiency of Tool Usage.

6.6 Summary

Working at the tactic level study of OSS adoption, Section 6 proposed a comprehensive assessment framework for OSS projects. This framework relies on the data from the public archives of OSS projects. Figure 7 shows a conceptual diagram of tactic level study of OSS adoption. Relevant propositions were formulated for future study. Empirical research questions were also posed in order to be answered in Section 7.

Figure 7: A conceptual diagram of tactic level study of OSS adoption

Tactic Level Study of OSS Adoption Assessment Framework for OSS Projects	
Managerial Analysis	Investigate the philosophical, legal, and financial issues that determine the behavioral regulations of the OSS project.
Goal Analysis	Determine if the objectives and methodology of the OSS project meet the requirements of the adopter.
License Analysis	Determine if the licenses adopted by the OSS project can meet the requirements of the adopter.
Value Analysis	Assess the financial feasibility and profitability of the adoption of the OSS project.
Technical Analysis	Scrutinize the technical facets of the products produced by the OSS project.
Execution Analysis	Assess the functionality, performance, usability, and other quality features of OSS products by execution.
Source Code Analysis	Analyze the quality of source codes of the OSS products.
Evolution Analysis	Investigate the evolution of the OSS project.
Supportive Analysis	Examine the vitality of the OSS community and the support and help that can be obtained from the OSS community.
Community Analysis	Assess the vitality of the OSS community by studying the behaviors of the members of the community.
Documentation Analysis	Assess the quality and quantity of design documents and user documents in the OSS projects.
Tool Usage Analysis	Investigate the usage of various tools in the OSS project.

7 Empirical Data Analysis

As stated in Section 1, one important component of this thesis is to analyze comprehensively the public archives of a rich set of OSS projects, which have not been studied in previous studies. This section conducts an empirical data analysis of a large number of OSS projects. The empirical research questions posed in Section 6 are answered in this section by summarizing the assorted attributes of OSS projects based on data collected from the public archives. The description of data used in this analysis is first introduced, followed by detailed data analysis in the following sections.

7.1 Description of Data

This section describes the data sets, data collection, and contents of data used in the empirical data analysis.

7.1.1 Data Sets

OSS projects which use Python as one of their programming languages (Python projects) are the focus of the empirical data analysis in this thesis. As indicated in Section 4.3.1, this approach can fathom comprehensively the real world usage of Python. In addition to considerations from the perspective of the impacts of programming languages, Python projects are chosen because as a group they have not been studied in previous research and they ensure a sufficient study size.

The data source of this data analysis is SourceForge.net. Well known by the OSS community, SourceForge.net claims that it is the world's largest OSS development site (http: // sourceforge.net). An overview of the services provided by SourceForge.net is

attached in Appendix B. It is also the primary data source for several previous quantitative OSS studies (e.g., Crowston & Scozzi, 2002; Krishnamurthy 2002; Madey et al., 2002a; Madey et al., 2002b), whose authors consider the projects at SourceForge.net are representative of the overall OSS projects due to the popularity of SourceForge.net and the large quantity of projects and developers registered there. In another word, it is difficult to find any other OSS repository better than SourceForge.net. Although some high profile OSS projects (e.g., Linux, Apache, Perl, etc.) are not hosted at SourceForge.net (Madey et al., 2002a; Madey et al., 2002b), this deficiency can be compensated by existing or future case studies of those high profile projects.

According to the classification proposed in Section 2.2, this empirical data analysis is a Population Project Study that obtains and analyzes data from a specific population of OSS projects. In this case, the specific population is all Python projects hosted at SourceForge.net. It is then called the Python Project Set (PPS).

Although the PPS is a kind of population by itself, it is just a subset of a bigger population – all OSS projects hosted at SourceForge.net. This larger population can be called the SourceForge Set (SFS). In this sense, the PPS can be considered as a sample from the SFS. Of course, the SFS itself is a subset of an even bigger population – all OSS projects in the world. As mentioned above, the SFS can be deemed as a good representative of overall OSS projects.

In order to depict the general characteristics of SFS and to test if the PPS is a good representative of SFS, summary data of several attributes of all project hosted at SourceForge.net are also collected and analyzed. This kind of data collection and analysis resemble the collection and analysis of demographic data for the population in a human

related sampling study. Then attributes of the sample (the PPS) are compared to the collected attributes of the population (the SFS).

7.1.2 Data Collection

Data collection in some previous quantitative studies of OSS projects was carried out manually by browsing project web sites at SourceForge.net and noting down the pertinent data. For instance, this was the way Krishnamurthy (2002) used to collect data from 100 projects. However, this manual method would be too time-consuming and error-prone for this thesis which aims to collect more extensive data from a much larger number of projects.

Software tools such as intelligent agent were used in attempts to retrieve data for this thesis from the web site of SourceForge.net. However, this kind of data collection method is constrained by two factors: power and cost of the tool, and structure of the target website.

On the tool side, no suitable OSS tool was found after thorough searching. As the consequence, proprietary closed source tools had to be tested. Two promising tools were chosen as candidates: NetStepper (http: //www.webattack.com) and Network Query Language (http: //www.e-botz.com/). Unfortunately, a dilemma was encountered. On one hand, affordable tools like the NetStepper and the student version of Network Query Language are not powerful enough for the task of data collection for this thesis. On the other hand, powerful tools such as other versions of Network Query Language are too expensive for a student (ranging from US $1000 to $100,000).

There are two caveats concerning our testing of tools. First, as described below, the power of tools is under the constraints posed by the other factor – the target website. Second, the testing of tools was not extensive. The evaluation version of NetStepper expired before producing any meaningful results. The student version of Network Query Language lacks the ability of web crawling by design. No wonder OSS is more preferable than these proprietary tools.

On the target website side, the SourceForge.net website does not provide semantic tags. Though it has some kind of structure that can be discerned visually by the bare eyes, it is hard to specify search patterns for an intelligent agent only according to the innate HTML presentation tags on the SourceForge.net website. Thus, the power of tools is at the mercy of the web designer of the target site.

Since SourceForge.net regularly collects statistics from the projects along with normal project data, it is far more convenient and efficient to extract data and statistics directly from the databases of SourceForge.net. Although under the pressure of limited resources and heavy workload, the support team of SourceForge.net upon request kindly offered help by extracting data from their databases.

Unfortunately, after several rounds of follow-up, the extracted data still cannot meet the requirements as originally planned. Consequently, the design and contents of this thesis have been adjusted based on the constraints of available data. The data file extracted on 3 October 2002 become the data source for the PPS. There are 1907 projects in the file. Thus the size of the PPS is 1907.

The most serious deficiency of the extracted PPS data set is the lack of historical data. All data are summary statistics as of the date of extraction. For example, there is

only the total number of messages posted to all forums of a project for the whole lifespan of the project. Ideally, the actual monthly volumes of each forum can depict the vivid fluctuation of information exchange along the evolution of the project. As a result, it is impossible to conduct time series analyses on the PPS. There are also missing data and errors which are explained in later appropriate sections.

For the SFS, the author of this thesis collected summary data for all projects from the website of SourceForge.net. These summary data are in the form of counts of projects classified by different criteria such as Development Status or Programming Language. The data collected on the same date (3 October 2002) of the final extracted data file for the PPS become the data source for the SFS.

According to the home page of SourceForge.net on 3 October 2002, there are 48,331 hosted projects, and 490,433 registered users. Thus, the size of the SFS is assumed to be 48,331. However, the above mentioned counts of projects classified by different criteria are provided without indication of the sample size (total number of all projects used in producing the counts). If all counts under a certain criteria are summed up, the resulted sample size is different from the number of 48, 331, and from sums under other criteria. For example, the sum of all projects classified by Development Status is 36,245, far less than 48,331. And the sum under Operating Systems is 44,871. Considering some projects may be redundantly counted because they have multiple values for a criterion (e.g., a project may indicate more than one development status), this symptom of the sum less than 48,331 is not reasonable.

About these inconsistencies among statistics, the support team of SourceForge.net simply answered that different arguments in SQL statements were used in generating

these counts. No details about the arguments were provided. As revealed in later sections of data analysis, some projects have missing values in some attributes and some projects have multiple values for some attributes. It is unknown how these missing values and multiple values are processed by SourceForge.net in producing the statistics on its web site. Nor is it known how dead projects or inactive projects are processed. These inconsistencies and uncertainties of the summary data provided on the web site of SourceForge.net severely restrict the value of the SFS as well as analyses that may be applied to the SFS. For example, even rudimentary descriptive statistics like percentage are not feasible due to the unknown size. Thus, the SFS should be interpreted with the limitations on mind, and be used only as a reference for the PPS.

7.1.3 Data Source File

The data source file of the PPS provided by SourceForge.net is in the format of "field::value(s)" pairs. All data in this file are as of 3 October 2002. The fields in the file can be classified into two classes: single value fields and multiple value fields. The former can only contain one single value per field. The latter may contain multiple values per field. The classification and description of the fields as well as example values of the fields are presented in Table 4.

The multiple value fields comprise a section called "trove" by SourceForge.net (Appendix B). As informed by SourceForge.net, information in this trove section is provided by the projects themselves. SourceForge.net does not validate these fields in any way. As a result, there are missing values in these fields since they are not mandatory as per the policy of SourForge.net.

Table 4: Fields and example values of the PPS data source file

Field	Example Value	Comment
Single Value Fields		
Numeric identifier of the project	5470	Unique within the PPS
A word for the name of the project	python	Unique within the PPS
Date of registration at SourceForge.net	2000-May-08	Provided in the format of Unix time
Number of administrators	5	
Number of developers who are not administrators	41	
Number of releases	38	Not correctly extracted
License used	python	
Number of mailing lists	3	
Number of forums	5	
Number of messages in the forums	2	
Number of bugs reported in the Bug Tracker	2898	
Number of open bugs	292	
Number of patches	1716	
Number of open patches	112	
Number of feature requests	131	
Number of open feature requests	90	
Number of support requests		Support Tracker is not used in the Python project
Number of open support requests		
Number of "add" into the CVS	808	Not correctly extracted
Number of "commit" into the CVS	808	Not correctly extracted
Number of on-line survey conducted	0	
Full name of the project	Python	
Homepage of the project	python.sourceforge.net/	
Short description (256 characters) of the project	The Python programming language, an object-oriented scripting and rapid application	
Multiple Value Fields		
Development Status	6 - Mature	
Environment	Console (Text Based); X11 Applications; Win32	
Intended Audience	End Users/Desktop; Developers; System	
License	OSI Approved	
Natural Language	English	
Operating System	OS Independent; Microsoft; POSIX	
Programming Language	Python; C; C++	
Topic	Software Development	

7.2 What types of applications do OSS projects provide?

There are several attributes from the PPS and the SFS that can describe what types of applications OSS projects provide. They are Topic, Intended Audience, Operating Systems, and Environment.

7.2.1 Topic

The attribute Topic for OSS projects at SourceForge.net refers to the application area of a project. The classification schema of topics and numbers of projects falling into each category, for PPS and SFS, are depicted in Table 5. Included in this table are also percentages for each category in the PPS.

Table 5: Topics in PPS and SFS

Topic	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	50	2.62%	-
Software Development	430	22.55%	6,356
Internet	424	22.23%	9,314
System	323	16.94%	7,535
Communications	291	15.26%	6,009
Games/Entertainment	263	13.79%	5,677
Scientific/Engineering	243	12.74%	3,001
Multimedia	228	11.96%	4,522
Database	128	6.71%	2,368
Office/Business	121	6.35%	1,768
Desktop Environment	96	5.03%	1,401
Education	81	4.25%	1,109
Text Editors	77	4.04%	1,043
Other/Nonlisted Topic	65	3.41%	1,002
Security	55	2.88%	1,058
Terminals	16	0.84%	237
Printing	9	0.47%	164
Sociology	5	0.26%	117
Religion	4	0.21%	103

OSS projects seem to be omnipresent in all kinds of application areas, even in unusual areas like religion. There are 2.62% projects in the PPS which do not indicate their topics, while this kind of projects are not identified in the SFS. As revealed in the tables of later sections, this ignorance of missing data is omnipresent in the SFS. Thus, it will not be mentioned again in later sections. Projects in the PPS cover the full spectrum of topics in the SFS. There are only slight differences in the order of popularity of topics between PPS and SFS. The relative precedence among the top 3 topics (Software Development, Internet, System) varies between PPS and SFS. And Scientific / Engineering switches place with Multimedia between PPS and SFS.

The counts of projects by number of topics for PPS are shown in Table 6. The majority of projects involve one to three topics. Only 43 projects involve more than three topics.

Table 6: Counts of projects by number of topics in PPS

Number of Topics	Count of Projects	Percentage
Not Indicated	50	2.62%
1	893	46.83%
2	571	29.94%
3	350	18.35%
4	32	1.68%
5	8	0.42%
6	3	0.16%
Total	1907	100.00%

7.2.2 Intended Audience

The attribute "Intended Audience" can expose the purpose of the project. The classification schema of intended audiences and numbers of projects falling into each category, for both PPS and SFS, are depicted in Table 7. Included in this table are also percentages for each category in the PPS.

Table 7: Intended audiences in PPS and SFS

Intended Audience	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	43	2.25%	-
Developers	1,329	69.69%	20,129
End Users/Desktop	974	51.07%	17,573
System Administrators	477	25.01%	8,668
Other Audience	247	12.95%	4,463
Information Technology	8	0.42%	237
Education	8	0.42%	146
Science/Research	5	0.26%	133
Telecommunications Industry	2	0.10%	46
Customer Service	1	0.05%	38
Healthcare Industry	1	0.05%	16
Manufacturing	0	0.00%	17
Financial and Insurance Industry	0	0.00%	16
Religion	0	0.00%	10
Legal Industry	0	0.00%	7

Obviously, the classification scheme of intended audience adopted by SourceForge.net is a blend of two different concepts: target users and target application areas. This kind of blend seems not to be accepted by the OSS community. So the majority of projects just indicate target users, as shown in Table 7. And there are 2.25% project in the PPS do not indicate their intended audience.

OSS projects cater to the needs of quite divergent audiences. PPS covers the majority of the spectrum of intended audiences in the SFS, with exceptions for some uncommon audiences such as religion and legal industry. Although the other two missing audiences (manufacturing, and financial and insurance industry) are quite important industries, they do not get much attention in the SFS either.

The counts of projects by number of intended audiences in the PPS are shown in Table 8. Less than half projects only target at single audience, while the other half target at two to four audiences.

Table 8: Counts of projects by number of intended audiences in PPS

Number of Types of Intended Audiences	Count of Projects	Percentage
Not Indicated	43	2.25%
1	926	48.56%
2	701	36.76%
3	224	11.75%
4	13	0.68%
Total	1907	100.00%

7.2.3 Operating System

The usage of Operating Systems in PPS and SFS is depicted in Table 9. About 6% projects in the PPS do not indicate the operating systems under which their products can run. The order of popularity of all operating systems is the same for both PPS and SFS. PPS covers the full spectrum of named operating systems used in SFS, though information about the operating systems under the category "Other OS" is not available.

Table 9: Usage of operating systems in PPS and SFS

Operating System	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	115	6.03%	-
POSIX	1,110	58.21%	19,909
OS Independent	756	39.64%	11,484
Microsoft	517	27.11%	10,698
MacOS	107	5.61%	1,359
Other OS	31	1.63%	630
PDA Systems	21	1.10%	419
BeOS	20	1.05%	299
OS/2	6	0.31%	73

PPS, like SFS, have strong heritage of Unix since the most popular operating system category – POSIX, conventionally refers to Unix-like operating systems including the famous GNU / Linux. However, POSIX is an international protocol for operating systems. It should be noted that Windows NT and other Microsoft OS are also POSIX compliant. So using POSIX as a category name may cause confusion.

There is an error in the PPS. Eighteen projects indicate their operating system as "PalmOS", where this category does not exist in the SFS. Thus "PalmOS" is combined to "PDA Systems" for the PPS, resulting in 21 projects in "PDA Systems".

One serious problem with Table 9 is that "OS Independent" is treated like just one kind of operating system. But logically, it should be treated as mutually exclusive to all other categories. And in reality, some projects also indicate other operating systems besides claiming "OS Independent". This can be considered as a logic error that can mask the feature of "OS Independent". If those projects select all other operating systems, then the effect of redundant counting can be cancelled out since every category gets one redundant count. However, it is not the case in the PPS.

The reason to keep Table 9 is because there is no way to cull "OS Independent" projects from the SFS. Fortunately, the PPS can be purified. For all projects which claim "OS Independent" as one of their operating systems, their indication of other operating systems are ignored. In another word, those "OS Independent" projects are separated from the counting of other categories. This solves the redundant count problem. The results after this manipulation are shown in Table 10.

Comparing Table 9 and Table 10, we can see that there are projects in every category (other than "Not Indicated" and "OS Independent") that make the logical error by selecting both "OS Independent" and other categories. Moreover, these projects just select some of the other operating systems, not all of the categories. This can be discerned from the different counts of these projects in different categories. For example, there is only one such project selected "OS/2" besides "OS Dependent", while there are 137 projects selected "POSIX" besides "OS Dependent". Python is a cross platform

programming language. It can support almost all popular operating systems. It is then a little surprise that only about 40% Python projects (Table 10) consider themselves "OS independent".

Table 10: Usage of operating systems in PPS, separating "OS Independent" projects

Operating System	PPS	
	Number	% (n = 1907)
Not Indicated	115	6.03%
POSIX	973	51.02%
OS Independent	756	39.64%
Microsoft	431	22.60%
MacOS	94	4.93%
Other OS	26	1.36%
PDA Systems	15	0.79%
BeOS	16	0.84%
OS/2	5	0.26%

The counts of projects by number of operating systems in the PPS are shown in Table 11. It should be noted that the projects in Table 11 do not include the two exclusive categories: "Not Indicated" and "OS Independent". Surprisingly, 61% of these projects support only one operating system. Only 6 projects support more than 3 operating systems.

Table 11: Counts of projects by number of operating systems in PPS

Number of Operating Systems	Count of Projects	Percentage
1	632	61.00%
2	291	28.09%
3	107	10.33%
4	5	0.48%
5	1	0.10%
Total	1036	100.00%

The unpopularity of "OS Independent" projects in the PPS could be considered as a shortcoming since OS independence is an advantage bestowed by Python. There are

several possible causes of this phenomenon. First, the developers of Python projects may focus on just one or two major target operating systems (as shown in Table 11), though their codes could run under other operating systems as well. Second, Python is a good glue language to incorporate modules written in other programming languages. Perhaps, those modules are more OS specific. Further study into individual projects may reveal other reasons.

7.2.4 Environment

The attribute "Environment" used at SourceForge.net seems to be interwoven with another attribute "Operating System". The classification and usage of environments in PPS and SFS are shown in Table 12. Three out of eight categories (the "Not Indicated" category is added by me) are tightly related to operating systems: X11 Applications, Win32, and Cocoa. And 14.63% projects in PPS do not indicate their environments.

Table 12: Environments in PPS and SFS

Environment	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	279	14.63%	-
X11 Applications	658	34.50%	8,134
Console (Text Based)	641	33.61%	8,490
Web Environment	469	24.59%	9,184
Win32 (MS Windows)	438	22.97%	7,171
Other Environment	227	11.90%	3,721
No Input/Output (Daemon)	187	9.81%	2,819
Cocoa (MacOS X)	26	1.36%	440
Handhelds/PDA's	12	0.63%	222

There are slight differences in the order of popularity of environments between PPS and SFS. The most significant difference is in Web Environment, which is No. 1 in SFS but No. 3 in PPS. The relative lower priority of Web Environment in PPS could be deemed as a shortcoming. Python is good at web programming. And a killer application

in Python – Zope – is an advanced application server that runs in web environment. The other difference is the relative precedence between X11 Applications and Console. It can be inferred from Table 12 that Python projects are still influenced by the heritage of Unix since the top 2 environments (X11 Applications and Console) are of strong Unix flavor.

The counts of projects by number of environments in the PPS are shown in Table 13. The majority of projects run in one to three environments. Only 22 projects run in more than three environments.

Table 13: Counts of projects by number of environments in PPS

Number of Environments	Count of Projects	Percentage
Not Indicated	279	14.63%
1	890	46.67%
2	463	24.28%
3	253	13.27%
4	15	0.79%
5	5	0.26%
6	2	0.10%
Total	1907	100.00%

7.3 What kinds of licenses are used in OSS projects?

There are two fields in the PPS that provide information about the licenses used by projects. One is a single-value field. The other is a multi-value field in the trove section. And these two fields use slightly different terms for describing licenses. This duplication adds unnecessary complexity to data analysis. There are 52 projects which do not provide any data in the multi-value license field, while there is no missing value in the single-value field. But the multi-value field is selected as the source for analysis on licenses. There are two reasons for this choice.

First, as argued in Section 6.3.2, the adoption of multiple licenses is a smart strategy for some OSS projects. Second, normal visitors to the web site of SourceForge.net can only see license information in the trove section of a project. For those 52 projects with missing multi-value license field, visitors cannot find license information on the web site either.

The usage of licenses in PPS and SFS are shown in Table 14. There are 2.73% projects in the PPS which do not indicate their licenses. Naturally, the majority of projects in both PPS and SFS adopt "OSI Approved" licenses, which are licenses deemed by the Open Source Initiative (OSI) as compliant to the Open Source Definition (Appendix A). Surprisingly, there are still projects using Other / Proprietary License although SourceForge.net aims to serve the OSS community.

Table 14: Usage of licenses in PPS and SFS

License	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	52	2.73%	-
OSI Approved	1,798	94.28%	31,210
Public Domain	43	2.25%	985
Other/Proprietary License	33	1.73%	617

There are a few projects (2.25% in the PPS, and 985 in the SFS) which are in the Public Domain. This kind of license strategy, actually a no-license-at-all strategy, should be discouraged to the OSS community since it provides no protection at all. Anyone can easily take a public domain program private (DiBona et al., 1999). So it is the easiest prey of Microsoft's "embracing, extending, and extinguishing" maneuver. Though not explicitly against the use of Public Domain in OSS, Perens (DiBona et al., 1999) suggested that a developer should consider applying her own copyright to a public

domain program and re-licensing it if she is doing significant work on it. It would be interesting to investigate the minority of OSS projects who adopted Public Domain. We may find out the reasons for this choice and the real impact of this license strategy on these projects. One hypothesis is that since public domain provides no legal protection at all, not many developers are willing to contribute to these projects.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 26: There will be fewer developers who are willing to contribute to projects under public domain than developers who are willing to contribute to projects under OSI approved licenses.

The counts of projects by number of licenses in the PPS are shown in Table 15. The majority of projects (92.34%) just adopt one license. So dual / multiple license strategy is not yet widely applied by OSS projects. However, there are still 94 projects adopted this strategy. Thus, inspection of these projects who adopted dual / multiple license strategy may reveal if this strategy is a smart choice. As discussed in Section 6.3.2.2, Partners may prefer these OSS projects.

Table 15: Counts of projects by number of licenses in PPS

Number of Licenses Used	Count of Projects	Percentage
Not Indicated	52	2.73%
1	1761	92.34%
2	81	4.25%
3	12	0.63%
4	1	0.05%
Total	1907	100.00%

7.4 What kinds of programming languages are used in OSS projects?

The classification of programming languages and their usage at SourceForge.net are retrieved from the web site of SourForge.net and shown in Table 16. Although 41 named programming languages are actually used in projects, only 23 (56%) of these languages are used in more than 100 projects. Moreover, only 7 (17%) of these languages are used in more than 1000 projects. According to Table 16, Python is the 6th most popular programming language at SourceForge.net. Thus, Python projects could be a good representative for all projects at SourceForge.net. Not at the extremes, they are like the median in the spectrum of projects in terms of programming languages. Furthermore, they are large enough to ensure proper sample size.

However, there are a few caveats concerning the data in Table 16. One obvious error in Table 16 is that there are two "Assembly" items. Since some projects use more than one programming languages, it is unknown how SourForge.net counted these projects in Table 16. It is also unknown if inactive or dead projects were included in the counting. As a result, the count of 1881 for Python in Table 16 is different from the 1907 projects in the extracted data source file of the PPS. Thus, the calculation of percentage for each language across projects is meaningless based on Table 16 since the sample size is unknown. This is a vivid example of the problems of inconsistency and uncertainty in the SFS, as pinpointed in Section 7.1.2.

Notably, Zope is not really a programming language. Written mostly in Python, Zope is a leading open source application server, specializing in content management, portals, and custom applications (http: // www.zope.org). Considered widely as a killer

application in Python, Zope enables teams to collaborate in the creation and management of dynamic web-based business applications such as intranets and portals.

Table 16: Usage of programming languages in SFS

No.	Programming Language	Count	No.	Programming Language	Count
1	C	8,955	23	Object Pascal	113
2	C++	8,277	24	ML	86
3	Java	6,546	25	Zope	80
4	PHP	4,935	26	Fortran	71
5	Perl	3,842	27	Cold Fusion	64
6	Python	1,881	28	Prolog	51
7	Visual Basic	1,002	29	Eiffel	50
8	Unix Shell	909	30	Ada	49
9	JavaScript	848	31	Forth	45
10	Assembly	792	32	Smalltalk	38
11	Delphi/Kylix	774	33	Rexx	20
12	Other	599	34	Erlang	19
13	PL/SQL	569	35	Xbasic	17
14	Tcl	568	36	PROGRESS	13
15	C#	380	37	Pike	11
16	ASP	282	38	REBOL	11
17	Objective C	228	39	APL	6
18	Lisp	206	40	Euphoria	6
19	Pascal	189	41	Logo	5
20	Ruby	166	42	Modula	1
21	Assembly	144	43	Euler	0
22	Scheme	125	44	Simula	0

The counts of projects by number of programming languages in the PPS are depicted in Table 17. Evidently, more than half projects (966 / 50.66%) utilize only one programming language – Python. For these projects, Python is powerful enough to fulfill their objectives. However, other projects add other programming languages to their arsenal. The majority of them just add one or two other languages. But there are also a few projects (50) deploy from 3 to 5 languages other than Python.

Table 17: Counts of projects by number of programming languages in PPS

Number of Programming Languages Used	Count of Projects	Percentage
1	966	50.66%
2	581	30.47%
3	310	16.26%
4	26	1.36%
5	13	0.68%
6	11	0.58%
Total	1907	100.00%

The relationship between Python and other languages in these multiple-language projects can only be determined after individual investigation into the design details of specific project. The common sense in Python community (Brown 2001; Chun 2001; Lutz 2001) is to use Python for prototyping and the majority of implementation, and to use C or C++ for critical parts that need high execution speed. Is this common sense true in practice? Though individual investigation of projects is still needed, the usage counts of other programming languages as shown in Table 18 can give a hint.

It is interesting that the top 5 languages in Table 16 are almost identical to the top 5 languages in Table 18. Only the top 10 languages in Table 18 are used in more than 1% Python projects. As expected, C and C++ are the two most popular languages used together with Python. They are perfect complements to Python. It is not a surprise that Java is the third popular language. There is a special kind of Python implementation called Jython (http: //www.jython.org) which is written in 100% pure Java. On one hand, Jython can compile Python code into Java byte code. So Jython program can directly deploy the rich Java library and run on any Java platform. On the other hand, Jython provide useful functions such as scripting, interactive interpreter, and rapid application development to Java. Thus, Jython and Java work seamlessly in tandem.

Table 18: Usage of other programming languages in PPS

No.	Programming Language	Count	% (n = 1907)	No.	Programming Language	Count	% (n = 1907)
1	C	370	19.40%	17	Delphi/Kylix	6	0.31%
2	C++	341	17.88%	18	Eiffel	6	0.31%
3	Java	183	9.60%	19	Objective C	6	0.31%
4	Perl	116	6.08%	20	C#	5	0.26%
5	PHP	81	4.25%	21	Pascal	5	0.26%
6	Zope	70	3.67%	22	Prolog	4	0.21%
7	Unix Shell	40	2.10%	23	ASP	3	0.16%
8	Tcl	32	1.68%	24	Forth	2	0.10%
9	JavaScript	23	1.21%	25	ML	2	0.10%
10	PL/SQL	19	1.00%	26	REBOL	2	0.10%
11	Assembly	16	0.84%	27	Smalltalk	2	0.10%
12	Scheme	13	0.68%	28	Ada	1	0.05%
13	Ruby	10	0.52%	29	Cold Fusion	1	0.05%
14	Fortran	8	0.42%	30	Euphoria	1	0.05%
15	Lisp	8	0.42%	31	Object Pascal	1	0.05%
16	Visual Basic	8	0.42%	32	PROGRESS	1	0.05%

It is a surprise that Perl and Tcl come to the top part of the list since they are more or less direct competitors to Python. Again, individual investigation into specific projects is required to understand the real reason. However, a peek at some projects (e.g., http: //sourceforge.net /projects /wxwindows/) reveals that Python, Perl, or Tcl are just used to provide a variety of interfaces. Other low-level languages such as C and C++ are the workhorse in these projects. As convenient add-ons to those low-level languages, Python, Perl, or Tcl are just optional parts, not the core part. They are not directly competing with each other for implementing the major functionality of the projects.

The other three top languages, PHP, Zope, and JavaScript, are all web oriented languages or environment. From this fact, it can be stated that Python is very useful in web programming. For other programming languages with more or less frequency, it is amazing that Python can cooperate with such diverse cohorts. Though less than 42 as in the case of SFS, 32 programming languages are still a rich set for the PPS.

Figure 8: Emergence of Python Projects over Time

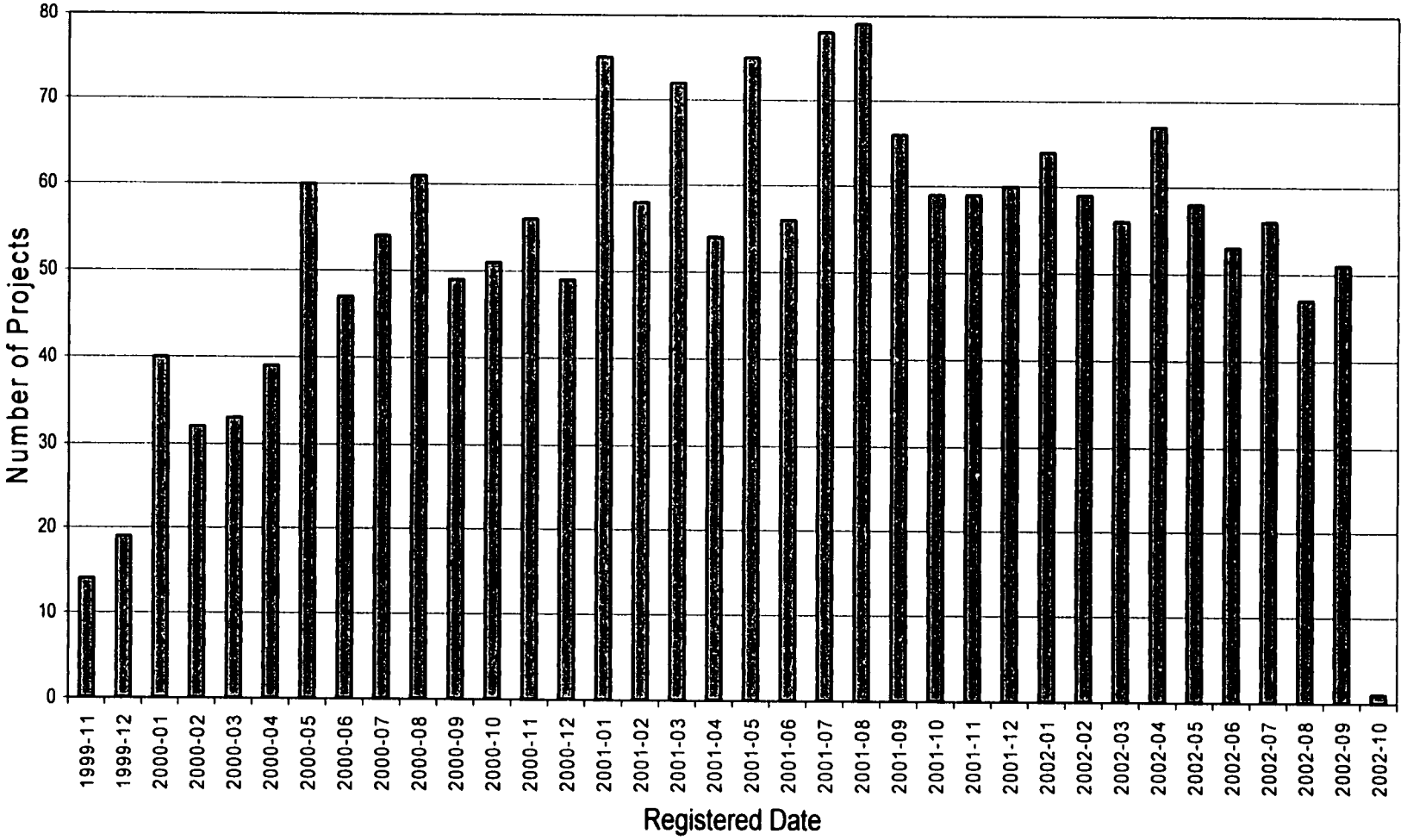


Figure 9: Development Status of Python Projects over Time

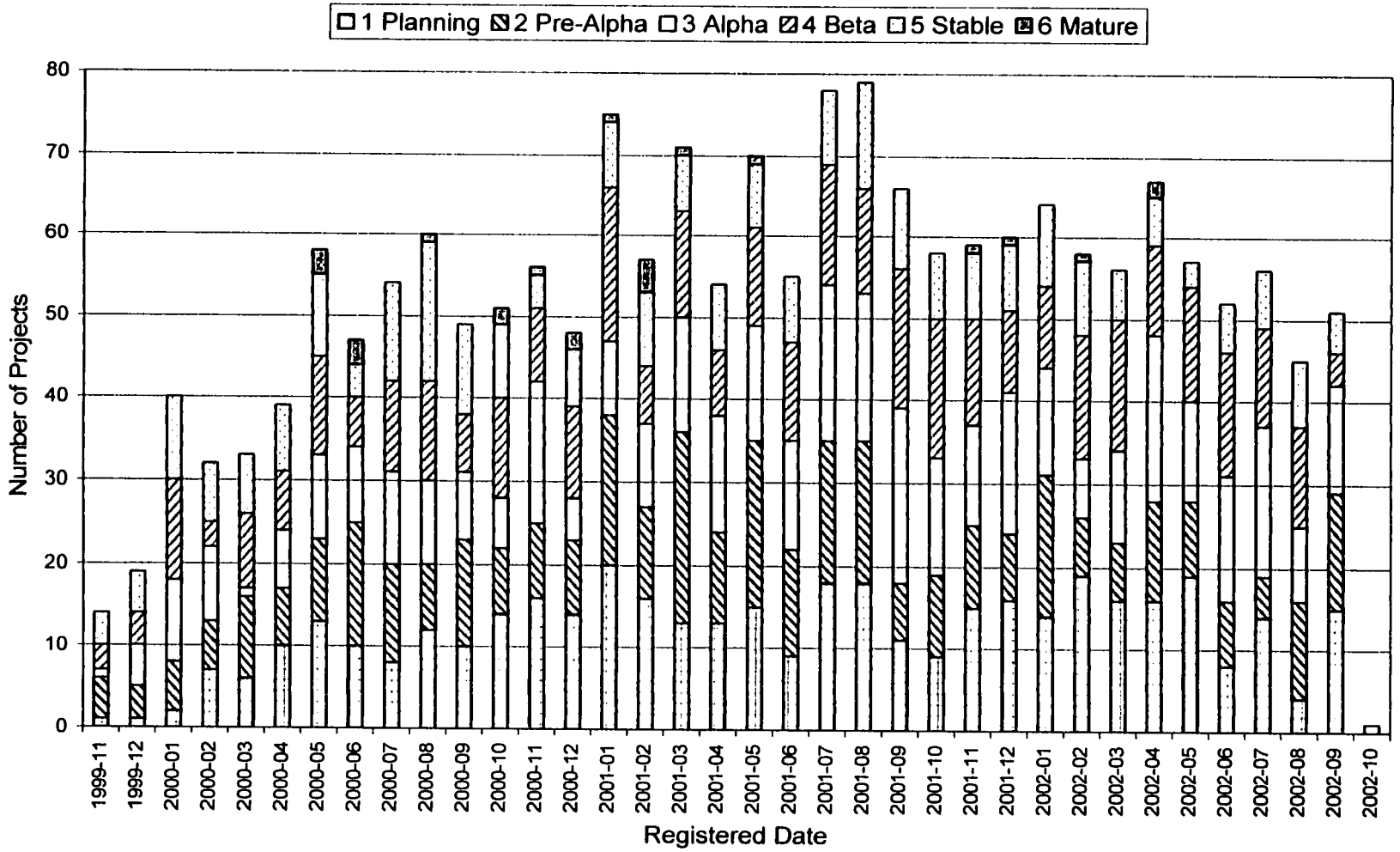
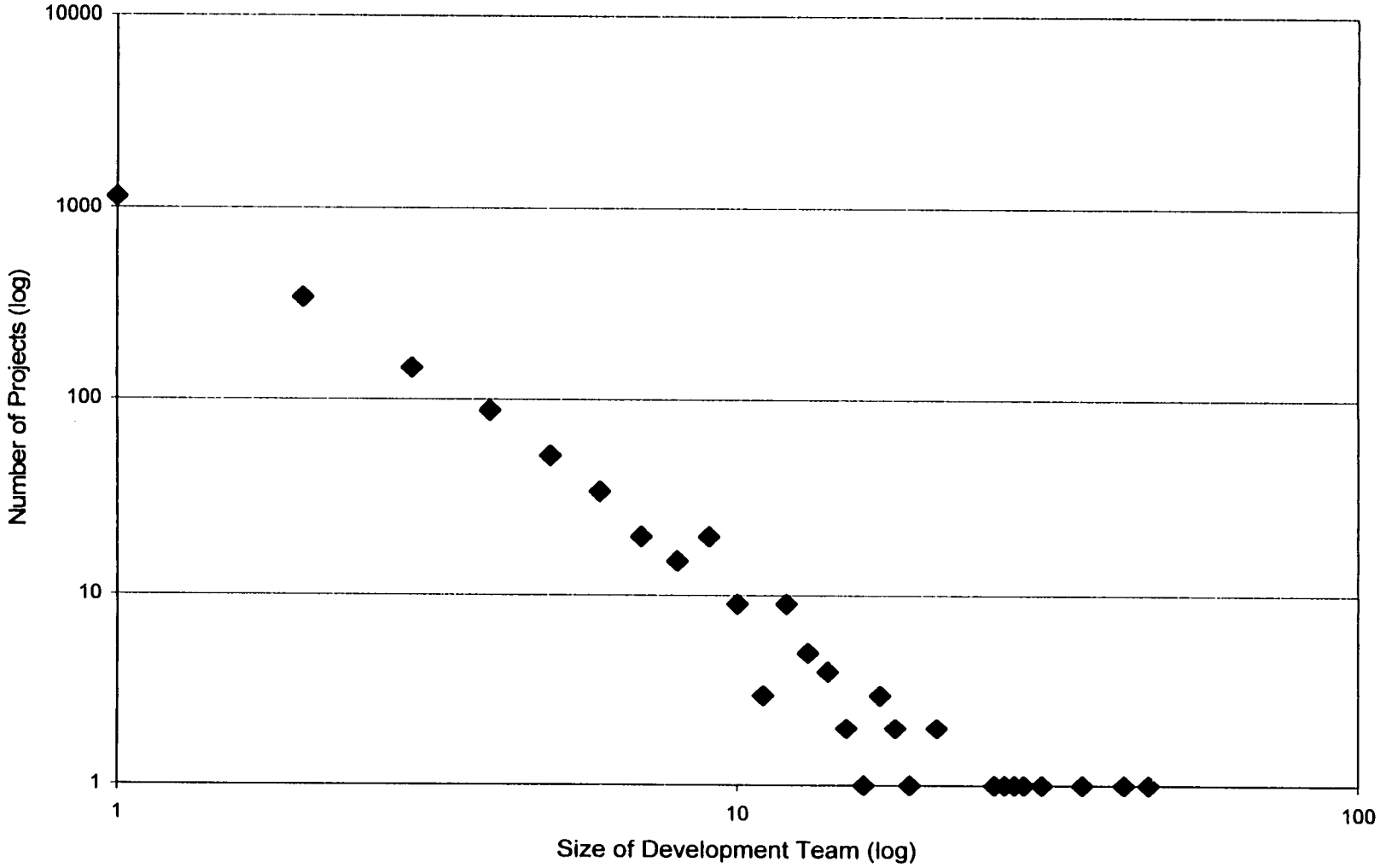


Figure 10: Power-law relationship for the size of project teams



7.5 What are the characteristics of the evolution of OSS projects?

The characteristics of the evolution of OSS projects can be depicted using data from the PPS and SFS. Specifically, we may look at the emergence of Python projects, the Development Status of PPS and SFS, and releases of the PPS.

7.5.1 Emergence of Python Projects

As seen from Figure 8 (p. 139), the emergence rate of new Python projects (counted as new projects registered at SourceForge.net) fluctuates over the three–year period (November 1999 – October 2002). However, it remains at the rate of over 1 new project per day (on average) after the initial two months, and retains a rate of over 40 projects per month after the initial 6 months. It should be noted that the data for October 2002 is only until October 3.

7.5.2 Development Status

OSS projects hosted at SourceForge.net are classified into 7 categories according to their development status. The classification schema and numbers of projects falling into each category, for both PPS and SFS, are depicted in Table 19. Included in this table are also percentages for each category in PPS.

It should be noted that it is the developers of an OSS project who judge the development stage or status that the project is currently in. Only administrators of projects can assign or change their projects' development status. Notably, the 7^{th} status "Inactive" was recently added in September 2002. So not too many projects have the chance to use this new status.

Table 19: Development Status in PPS and SFS

Development Status	PPS		SFS
	Number	Percentage	Number
0 - Not Indicated	17	0.89%	-
1 - Planning	422	22.13%	10,059
2 - Pre-Alpha	378	19.82%	6,764
3 - Alpha	403	21.13%	6,096
4 - Beta	383	20.08%	7,214
5 - Production/Stable	279	14.63%	5,492
6 - Mature	24	1.26%	593
7 - Inactive	1	0.05%	27
Total:	1,907	100%	36,245

Some projects in the PPS have more than one development status; there are 179 projects which have two statuses, and 15 projects which have 3 statuses. These multiple statuses may mean that some components of the project reached the status of, for example, stable, while the others are still at the stage of beta. However, the real reason for each project can only be certain after investigating individually the concerned projects. Since the unit of analysis of this thesis is a project as a whole, the highest status (1-7) was chosen as the single development status for a project with multiple statuses. Accordingly, the counts for Python projects in Table 19 are generated using only single development status without redundant counting. It is unknown how SourForge.net counted those projects with multiple statuses in the SFS.

As stated before, development status, like other multiple value fields, is not a mandatory attribute to be maintained at SourceForge.net. This is a shortcoming of SourceForge.net since development status is an important attribute for assessing the project. On the contrary of those multi-status projects, 17 projects in the PPS do not indicate their development status.

A project could be volatile before it reaches the stable stage. It is a common practice in previous studies (e.g., Krishnamurthy 2002) that sampling is restricted to projects reached the stage of Mature. However, as shown in Table 19, the proportion of mature projects is too small (less than 2%) for both PPS and SFS. Thus, mature projects may not be representative of the majority of OSS projects.

An intuitive hypothesis about Development Status is that older projects may reach higher status. A project which emerged earlier (registered earlier at SourceForge.net) theoretically should have more chances and time to reach stable or mature stages than a newly emerged project. The development status of Python projects over time is shown in Figure 9 (p. 140). The inactive projects (stage 7) and projects which do not indicate their status are removed from Figure 9. It is interesting that this intuitive hypothesis is proved to be wrong. For every month since the inception (except for October 2002, which only has 3-day data), the first 5 stages (from planning to stable) occur to some projects emerged in that month. On one extreme, some projects which emerged earliest in November 1999 still remain at the planning stage. On the other extreme, some projects which just emerged in September 2002 have reached the stable stage. Moreover, the highest stage – mature – occurs only to projects emerged in certain months. Notably, none of the projects emerged in the first half year period (November 1999 – April 2000) reached mature stage.

Thus, the age of project is not a good indicator of the project's development status. There are some possible explanations to the phenomenon shown in Figure 9. First, the objectives of the project, size of development team, and other factors have more influences to the development status of the project than the pure time effect. Second,

some projects have been well developed outside of SourceForge.net and they joined SourceForge.net later. So they may appear to reach higher stages fast. Third, some projects may actually remain inactive without changing their status to inactive. Further investigation into individual projects may reveal more reasons.

7.5.3 Releases

Examination of some randomly selected projects found that the values of the attribute "Number of Releases" in the extracted data file of PPS were not consistent with the statistics on the web sites of those projects at SourceForge.net. The support staff of SourceForge.net were consulted, but they did not provide valid answer due to resource limitation. Then the only feasible statement made out of those release statistics is that at least 835 (43.79%) projects did not post any release at SourceForge.net.

There are two comments about this statement. First, it should be noted that some projects maintain their own web sites outside of SourForge.net. They (e.g., the Python project and the Jython project) may put their releases on these sites rather than publishing through SourceForge.net. So some of these 835 projects may behave in this way. Second, some projects have no release posted on their SourceForge.net web site, but there are values in their extracted "Number of Releases" field. For example, for the Python project, there are 38 releases as in the extracted file, while "no release" as indicated on its SourForge.net release page. That is why "at least" is added in the above statement.

Inconsistency among data is the biggest enemy for research based on a large sample since the researchers don't have the luxury to deal with specific individual project. It is really a pity that the attribute of "Number of Releases" in the PPS is almost futile for this thesis. The frequency and quantity of releases could be utilized as empirical evidences to

test the distinguishing characteristics of OSS – "Release early, release often", as discussed in Section 4.2.1 and Section 6.4.3. This can be a valuable topic for future research.

7.6 What are the characteristics of development teams in OSS projects?

The development team of a project hosted at SourceForge.net consists of two types of members – Project Administrator and Developer. Each project has one or more Project Administrators who are in charge of the project. Project Administrators are often the initiators of the hosted project or primary authors of the software produced in the project. They mainly play the roles of "Project Leader" and "Core Developer" in the community of this project. Developers are team members who do not have administrative privileges. A person who wants to become Developers of a project has to contact a Project Administrator of the project. It is the Project Administrator who can add a new member to the team. Developers may play the role of "Core Developer", "Central Developer", "Peripheral Developer", or "Active User", depending on their contributions.

The characteristics of development teams in OSS projects can be depicted using data from the PPS and SFS. Specifically, we may look at the team size and natural language issues.

7.6.1 Team Size

The counts of projects by number of team members in the PPS are shown in Table 20. Notably, there are 5 projects which do not have either administrator or developer. Checking the web site of SourceForge.net indicates that these 5 projects do not exist there. So these 5 projects can be considered as dead projects which only leave their

historical trace in the database of SourceForge.net. The rate of dead projects for the PPS seems to be quite low (0.26%).

Table 20: Counts of projects by number of team members in PPS

	Administrator		Developer		Team	
Number of Members	Count of Projects (Percentage)		Count of Projects (Percentage)		Count of Projects (Percentage)	
0	5	(0.26%)	1378	(72.26%)	5	(0.26%)
1	1451	(76.09%)	227	(11.90%)	1139	(59.73%)
2	313	(16.41%)	108	(5.66%)	340	(17.83%)
3	90	(4.72%)	49	(2.57%)	145	(7.60%)
4	26	(1.36%)	45	(2.36%)	88	(4.61%)
5	15	(0.79%)	20	(1.05%)	52	(2.73%)
> 5	7	(0.37%)	80	(4.20%)	138	(7.24%)
Total	1907	(100.00%)	1907	(100.00%)	1907	(100.00%)

Krishnamurthy (2002) found that the vast majority of 100 mature OSS projects were developed by a small number of individuals. Only 29% projects had more than 5 team members, while 22% had just one team member (i.e., the only administrator). And 51% projects just had one administrator.

From Table 20, the finding of Krishnamurthy is confirmed. The majority (59.73%) of project in the PPS are developed by one-person team. Only 7.24% projects have more than 5 team members. And 76.09% projects just had one administrator.

Since the number of projects with no developer (1378) is bigger than the number of one-person projects (1139), there are some projects (1378 – 1139 – 5 = 234, 12.27%) with more than one administrators but without developer. These "many administrators, no developer" projects are of interest to OSS researchers because of their peer-to-peer trait. Since all team members are administrators with same privileges, they may behave differently from the one-person teams, and from the teams with some developers who possess less privileges. Previous OSS studies did not recognize the possible value of

these "many administrators, no developer" projects. For example, Krishnamurthy did not notice the existence of this kind of projects since he did not calculate the statistics for the developers, just did that for administrators and the whole team.

Accordingly, development teams of OSS project can be classified into three categories: Single Team, Peer Team, and Hierarchical Team. A Single Team is a team that has only one member. In our case of OSS projects, a Single Team has only one administrator with no developers. A Peer Team is a team where all members have the same kind of privileges. In our case of OSS projects, a Peer Team has more than one administrator but without any developer. A Hierarchical Team is a team where some members have certain special privileges that are not bestowed to other members. In our case of OSS projects, a Hierarchical Team has at least one administrator as well as at least one developer. Administrators have more privileges than developers. Table 21 delineates the classification of development teams of OSS projects. Intuitively, Single Team is suitable for low complexity projects; Peer Team is suitable for medium complexity projects; and Hierarchical Team is suitable for high complexity projects.

Table 21: Classification of development teams of OSS projects

Category	Administrator	Developer	Trait
Single Team	1	0	Only one member.
Peer Team	> 1	0	All members have the same privileges
Hierarchical Team	≥ 1	≥ 1	Administrators have more privileges

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 27: Single Team will be most appropriate for projects with low complexity.

Proposition 28: Peer Team will be most appropriate for projects with medium complexity.

Proposition 29: Hierarchical Team will be most appropriate for projects with high complexity.

The descriptive statistics for project teams in the PPS are shown in Table 22. These statistics all confirm that projects in the PPS do not have a big development team. The means are only 1.35, 0.91, and 2.27 for administrator, developer, and the team, respectively. And the standard deviations are not high. The minimum of "0" for Project Administrator is due to the 5 "dead" projects which have neither administrator nor developers. The maxima for administrator and developer are 13 and 41, while the maximum for team member is 46. So there are still projects with strong human resources, though not as strong as high profile projects such as Linux.

Table 22: Descriptive Statistics for Project Teams in PPS

(n = 1907)	Administrator	Developer	Team
Mean	1.35	0.91	2.27
Median	1	0	1
Mode	1	0	1
Minimum	0	0	0
Maximum	13	41	46
Std. Deviation	0.82	2.71	3.10

Madey et al. (2002a; 2002b) hypothesized that OSS development could be modeled as self-organizing, collaboration, social networks. These social networks have highly skewed distributions, which under a log-log transformation results in a linear relationship called a power-law relationship. The power-law model can be determined by ordinary least squares regression in log-log coordinates. After analyzing over 39,000 projects hosted at SourceForge.net, they found that both the project-size (number of developers on

the project) and the number of projects joined by a developer have power-law distributions. The adjusted R^2 of the regression for the project size is .93, while for the project-joined data is .97.

The distribution of number of projects with certain size of project team, i.e., the project size analysis suggested by Madey et al. (2002a; 2002b), is shown in Figure 10 (p. 141). Evidently, the power-law relationship exists in the PPS. The adjusted R^2 of the regression for this distribution is .940, with p value less than .000.

Another intuitive hypothesis is that older projects may have more developers. Krishnamurthy (2002) already refute this hypothesis, though he did it vaguely by correlating "Release Date" and number of developers. His choice of "Release Date" could be a typo for "Registered Date" since the former often refer to software releases which are usually more than one for a project. It is the unique registered date that indicates the age of the project. The correlation coefficient between the registered date and the number of administrators in the PPS is only -0.08, while the correlation coefficient between the registered date and the number of developers is only -0.14. Thus, as in the case of development status (Section 7.5.2), the age of a project has no significant impact on the size of development team.

7.6.2 Small Teams and Competing Projects

The phenomenon of small development teams, of which the majority are not really a team at all since they are Single Teams, should be fathomed since the power of OSS development model relies on the power of the OSS community, of which the development team is a critical component. Small may not always mean weak. However, under the circumstance of a small development team, the number of "eyeballs" of this

team is obviously not enough to fulfill the Linus's Law. So contributions of other members of the OSS community become paramount to the success of the project. Further study of the community of these Single Team projects can reveal whether other members of the community contribute a lot. But intuitively, people will hesitate to join the community of a Single Team project.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 30: Compared with Peer Team projects and Hierarchical Team projects, Single Team projects will generally have lower Vitality of Community.

The causes of this phenomenon of small development teams are another area to be fathomed. Although Madey et al. (2002a; 2002b) pointed that the power-law relationship could be one property of OSS development which is a self-organizing social network, they admitted that it was still an open research question: Why do these networks have power-law relationship?

This thesis suggests another possible reason: A developer may prefer starting a new OSS project by herself to joining an existing project. As shown in the case of Email client discussed in Section 4.3.1, there are so many projects that strive to furnish the same or comparable functionality. Since they are all hosted at SourceForge.net, it is not hard to find out if similar projects exist. So at least, some of these projects are started by developers who do not want to join the existing Email Client projects. Why did these developers aspire to launch a new project? This question should be a good topic for the research on the motivations of OSS developers. It is perhaps because of the ego of these

developers. Technical issues are less important here since the developer who thinks that she has a better solution can suggest her ideas to the existing project.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 31: Competing OSS projects will occur more frequently among small development team projects than among large development team projects.

The convenient services provided by organizations such as SourceForge.net may objectively foster the emergence of competing projects aimed for the same or comparable functionality. People can easily start and run a project at SourceForge.net. One interesting thing is that any proposal to start a new project is manually reviewed by a staff of SourceForge.net to determine if it can be launched. So it may be guessed that subjectively SourceForge.net encourages duplication of similar projects, possibly due to its purpose of remaining as the largest repository of OSS projects. This purpose can be inferred from its emphasis on number of hosted projects and number of registered users. But quality should be more important than quantity.

As stated before, competent developers are precious resources for OSS movement. Those competing projects are really a waste of these valuable resources – competent developers in specific and OSS community in general. It may be argued that these projects are in a form of benign competition. And competition is good for a Consumer. But if we think carefully, competition is not what a Consumer wants. It is the best software that a Consumer wants. Although there are quite a few operating systems produced in the OSS development model, such as Linux, OpenBSD, FreeBSD, and NetBSD, a Consumer really wants just one that is most suitable for her needs. So if she

chooses Linux, all other operating systems are just like dust for her. The efforts of developers of these OSS are just wasted for this Consumer. The same kind of waste happens to the case of Email client mentioned before. It is unlikely that a Consumer uses more than one Email clients. So the question of competition turns out to be really a question of how to produce the best software.

In the case of proprietary closed source software, there are usually fierce competitions from different development teams of different companies. Since design documents and source codes are kept as secrets of each team, these teams can only learn from each other indirectly, by means of Execution Analysis and reverse engineering. It is practically impossible to pool the wisdom of these competing teams together to produce the best solution. The fierce competition is a roundabout way, however, the only viable way, to produce better and best solutions.

OSS can change the rules of the game totally. Since design documents and source codes are publicly available, the wisdom of interested developers can be pooled together to produce the best solution. Developers can learn from each other, and help each other directly. No reverse engineering is required. It is the collaboration of the OSS community, not the competition, that will produce the best OSS products and provide the best OSS services. The gist of OSS should be collaboration, not competition.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 32: In order to produce the best OSS products and provide the best OSS services, collaboration within one project will be more cost-efficient than will competition among similar projects aimed for the same or comparable functionality.

There is not really sufficient or necessary reason to start competing projects, and to fight for precious resources. One rational reason for starting competing projects could be using different programming languages. As discussed in Section 4.3.1, programming languages have many impacts on projects. Since programming languages are like natural languages, programmers using different programming languages may not be able to communicate and collaborate in the same project. As discussed in Section 6.4.2, programming languages have important implications for the Prosumer, Profitor, and Partner. So a competing project using suitable programming languages may be useful for these kinds of adopters. However, even this reason is not valid from the Consumer's point of view. The Consumer simply needs the best software no matter which programming language is utilized. Thus, those competing projects which use the same programming language are the worst case of waste since they directly draw water from the same well – programmers using the same programming language.

As a byproduct of the preceding discussion, certain propositions can be put forward for future study:

Proposition 33: Competing OSS projects will more likely use different programming language sets than use identical programming language sets.

Another rational reason for starting competing projects could be meeting different requirements that cannot be accommodated in the existing project. Usually, the existing project has serious limitations due to legacy reasons. But then, these projects are not exactly competitors. The new project is a successor of the old project. It is a case of evolution in stead of competition. One example is the Subversion project (http: // subversion.tigris.org) which aims to replace the famous CVS.

A third rational reason could be the divergence in solutions and the difficulty to judge which solution is the best. Each solution can be tested in a competing project in order to discern the best solution from trial. But then, it is for all people's benefits to merge these projects when a winner has been proved. In reality, this is the primary reason of forking in OSS projects such as the competition between GNOME and KDE, two desktop environments for Linux. And it is difficult and may take long time to decide the winner. GNOME and KDE already start merging in the sense that applications developed for one environment can also be used in the other, and it is easy to switch these two environments in Linux. However, the two are still competing for precious resources and still being wasted by Linux users since they normally only use one kind of desktop though they may use applications from both.

In a word, since OSS can pool the wisdom of all developers and all members of the OSS community together, it can produce the best software through collaboration, without the need of competition among similar projects. Unlike closed source software, the competing OSS projects are a waste of precious resources. Again, the motto of Python can make sense here: "There should be one – and preferably only one – obvious way to do it".

Further research is required to check if there are lots of competing OSS projects, and to ask the developers of these projects why they insist on running their own projects. Research on those one-person teams could also be valuable. For example, it is interesting to investigate how big and complex a project can be for a one-person team to handle.

7.6.3 Natural Language

The attribute of natural language is more descriptive for the developers than for the software that those developers produce since internationalization is a conventional requirement for modern software. With the support of Unicode, software can handle all sorts of natural languages. This is true for Python projects since Python supports Unicode natively. Thus, the attribute of natural language at SourceForge.net can be assumed to describe the languages used among developers and in documentation. Table 23 depicts the usage of natural languages in PPS and SFS.

There are some shortcomings of the language classification provided by SourceForge.net. First, there is not a catch-all item "Other Language". So the list is theoretically incomplete. Second, dead language like Latin is included in the list. Third, only Chinese and Portuguese are classified further into dialects. Although Simplified Chinese and Traditional Chinese use different character sets, they do not differ in terms of syntax, semantics, and grammar.

The order of most popular natural languages (especially the top 6) is the same for both Python projects and all projects. The PPS utilizes 15 different natural languages. The SFS utilizes 23 more. Especially, Italian, which is the 7th popular language in SFS, is not spoken in PPS. One project in the SFS is in Esperanto, an artificial language.

About 30% Python projects do not indicate their natural languages. English is obviously the dominant language with over 70% coverage, which is virtually all the projects which indicate their natural languages. However, there are still 1.84% (100% - 28.05% - 70.11%) Python projects do not use English. This is an interesting

phenomenon. Further study into those non-English projects may reveal the reason for not using English as one of their natural languages.

Table 23: Usage of natural languages in PPS and SFS

Natural Language	PPS		SFS
	Number	% (n = 1907)	Number
Not Indicated	535	28.05%	-
English	1,337	70.11%	23,968
German	77	4.04%	2,341
French	47	2.46%	1,454
Spanish	43	2.25%	850
Russian	13	0.68%	381
Japanese	9	0.47%	248
Korean	5	0.26%	7
Portuguese (Brazilian)	3	0.16%	28
Norwegian	2	0.10%	4
Dutch	1	0.05%	24
Chinese (Simplified)	1	0.05%	9
Chinese (Traditional)	1	0.05%	7
Hungarian	1	0.05%	7
Portuguese	1	0.05%	4
Bulgarian	1	0.05%	3
Italian	0	0.00%	47
Swedish	0	0.00%	11
Czech	0	0.00%	10
Polish	0	0.00%	9
Catalan, Hebrew	0	0.00%	7
Danish, Turkish	0	0.00%	6
Ukranian	0	0.00%	5
Greek, Indonesian, Romanian	0	0.00%	4
Tamil, Thai, Vietnamese	0	0.00%	3
Afrikaans, Arabic, Croatian, Serbian, Slovak	0	0.00%	2
Esperanto, Finnish, Urdu	0	0.00%	1
Bengali, Bosnian, Hindi, Icelandic, Javanese, Latin, Latvian, Macedonian, Malay, Marathi, Panjabi, Persian,Slovenian, Telugu	0	0.00%	0

The counts of projects by number of natural languages in the PPS are shown in Table 24. The majority (71%) of projects use only one or two natural languages. However, there are still 12 projects use more than 2 languages.

Table 24: Counts of projects by number of natural languages in PPS

Number of Natural Languages	Count of Projects	Percentage
Not Indicated	535	28.05%
1	1219	63.92%
2	141	7.39%
3	9	0.47%
4	1	0.05%
5	2	0.10%
Total	1907	100.00%

7.7 What are the usages of various tools in OSS projects?

The usages of various tools in OSS projects can be depicted using data from the PPS and SFS. Specifically, we may look at the usage of mailing lists, forums, surveys, CVS, Bug Tracker, Patch Tracker, Feature Tracker, and Support Tracker.

7.7.1 Mailing Lists

Krishnamurthy (2002) found that on average the 100 mature OSS projects in his study each had 2 mailing lists, and 21 of them did not have even one. The descriptive statistics for number of mailing lists used in the PPS are shown in Table 25. The mean is only 0.79. And both the median and the mode are zero. However, the maximum is 22, somehow impressive. The counts of projects by number of mailing lists in the PPS are shown in Table 26. More than 57% projects do not use mailing lists at all. Another 39.8% projects only have one to three mailing lists. The rest less than 3% projects deploy more than three mailing lists.

Table 25: Descriptive Statistics for Mailing Lists in PPS

(n = 1907)	# of Mailing Lists
Mean	0.79
Median	0
Mode	0
Minimum	0
Maximum	22
Std. Deviation	1.26

It is a pity that in the extracted data file of the PPS, there is no statistics for messages and threads in the mailing lists. So it is impossible to know the volume of the mailing lists. One good sign is that there are still projects which may use the mailing lists a lot since they have a large quantity of mailing lists (10 and 22).

Table 26: Counts of projects by number of mailing lists in PPS

Number of Mailing Lists	Count of Projects	Percentage
0	1099	57.63%
1	403	21.13%
2	212	11.12%
3	144	7.55%
4	26	1.36%
5	16	0.84%
6	3	0.16%
7	1	0.05%
10	2	0.10%
22	1	0.05%
Total	1907	100.00%

7.7.2 Forums

Krishnamurthy (2002) found that on average the 100 mature OSS projects in his study each had 2 forums, and 33 of them did not have even one. The counts of projects by number of forums in the PPS are shown in Table 27. Only a few projects do not use forums at all. The majority (78.29%) of projects adopts 2 forums. Actually, 95.49% projects utilize one to four forums. Less than 2% projects deploy more than four forums.

Compared to the statistics of mailing lists, these statistics prove the assertion in Section 3.6 that forums could become the next prevailing communication tool.

Table 27: Counts of projects by number of forums in PPS

Number of Forums	Count of Projects	Percentage
0	49	2.57%
1	48	2.52%
2	1493	78.29%
3	210	11.01%
4	70	3.67%
5	17	0.89%
6	11	0.58%
7	6	0.31%
8	2	0.10%
15	1	0.05%
Total	1907	100.00%

Descriptive statistics for forums in the PPS are shown in Table 28. The mean of number of forums is 2.19, while the median and mode are both 2. The maximum number of forums is 15, not too impressive. The mean of messages is 11.61. However, the maximum of messages is 3535. So there is a great variation in the volumes of messages posted in the forums. This is supported by the large standard deviation (126.23).

Table 28: Descriptive Statistics for Forums in PPS

(n = 1907)	# of Forums	# of Messages	Monthly Messages
Mean	2.19	11.61	0.64
Median	2	2	0
Mode	2	2	0
Minimum	0	0	0
Maximum	15	3535	126.25
Std. Deviation	0.85	126.23	4.37

Since projects have different ages, it is wise to use the monthly volume as a more appropriate indicator of volume. This is also the suggestion of the metric in Section 6.5.3.3.2. The statistics of monthly messages reveal that actually the forums are seldom

used. The median and mode of monthly messages are 0, while the mean is only 0.64 and the standard deviation reduces to 4.37. Interestingly, the maximum monthly messages are 126.25. So still some projects use forums regularly. Krishnamurthy (2002) found that the vast majority of the 100 mature OSS projects had very few messages over the life time of the projects. However, he did not use the monthly measures.

7.7.3 Surveys

The counts of projects by number of surveys in the PPS are shown in Table 29. The vast majority (94.81%) of projects do not use surveys at all. The rest projects conduct one to three surveys.

Table 29: Counts of projects by number of surveys in PPS

Number of Surveys	Count of Projects	Percentage
0	1808	94.81%
1	87	4.56%
2	10	0.52%
3	2	0.10%
Total	1907	100.00%

7.7.4 CVS

There is an obvious error in the statistics for CVS in the extracted data file of PPS: the number of "CVS commits" is the same as the number of "CVS adds" for every project. Examination of some randomly selected projects found that these identical numbers were not consistent with the statistics on the web sites of those projects at SourceForge.net. The support staff of SourceForge.net were consulted, but they did not provide valid answer due to resource limitation. Then the only viable usage of those CVS statistics is to treat them as dichotomous variable. If there is a number greater than zero, CVS is used in the project. If the number of "CVS commits" (or "CVS adds", since they

are the same) is zero, CVS is not used in the project. The transformed result for the usage of CVS in the PPS is shown in Table 30. The majority (71.84%) of projects utilize the CVS. The rest 28.16% projects do not use the CVS.

Table 30: Usage of CVS in PPS

Use of CVS	Count of Projects	Percentage
No	537	28.16%
Yes	1370	71.84%
Total	1907	100.00%

7.7.5 Bug Tracker

The descriptive statistics for Bug Tracker in the PPS are shown in Table 31. The statistics for Bug Tracker are missing for 88 (4.61% of 1907) projects. The usage of Bug Tracker seems to be very low since the medians and modes are zero for both "all issues" and "open issues". And the means are also quite low for both "all issues" and "open issues". However, the maxima and the standard deviations are impressively high. Thus, there are still projects which utilize the Bug Tracker a lot. This usage pattern is confirmed by the monthly statistics. The mean of monthly issues (all issues) is only 0.35, with median and mode being zero. But the maximum and standard deviation are still high, comparing to the mean.

Table 31: Descriptive Statistics for Bug Tracker in PPS

(n = 1819; 88 Missing)	All Issues	Open Issues	Monthly Issues
Mean	7.79	1.42	0.35
Median	0	0	0
Mode	0	0	0
Minimum	0	0	0
Maximum	2898	292	99.93
Std. Deviation	84.01	10.39	3.12

7.7.6 Patch Tracker

Descriptive statistics for Patch Tracker in the PPS are shown in Table 32. The statistics for Patch Tracker are missing for 210 (11.01% of 1907) projects. The usage of Patch Tracker seems to be very low since the medians and modes are zero for both "all issues" and "open issues". And the means are also quite low for both "all issues" and "open issues". However, the maxima and the standard deviations are impressively high. Thus, there are still projects which utilize the Patch Tracker a lot. This usage pattern is confirmed by the monthly statistics. The mean of monthly issues (all issues) is only 0.07, with median and mode being zero. But the maximum and standard deviation are still high, comparing to the mean.

Table 32: Descriptive Statistics for Patch Tracker in PPS

(n = 1697; 210 Missing)	All Issues	Open Issues	Monthly Issues
Mean	2.07	0.29	0.07
Median	0	0	0
Mode	0	0	0
Minimum	0	0	0
Maximum	1716	112	59.17
Std. Deviation	45.35	3.43	1.50

7.7.7 Feature Tracker

Descriptive statistics for Feature Tracker in the PPS are shown in Table 33. The statistics for Feature Tracker are missing for 27 (1.42% of 1907) projects. The usage of Feature Tracker seems to be very low since the medians and modes are zero for both "all issues" and "open issues". And the means are also quite low for both "all issues" and "open issues". However, the maxima and the standard deviations are pretty high. Thus, there are still projects which utilize the Feature Tracker a lot. This usage pattern is confirmed by the monthly statistics. The mean of monthly issues (all issues) is only 0.07,

with median and mode being zero. But the maximum and standard deviation are still high, comparing to the mean.

Table 33: Descriptive Statistics for Feature Tracker in PPS

(n = 1880; 27 Missing)	All Issues	Open Issues	Monthly Issues
Mean	1.19	0.67	0.07
Median	0	0	0
Mode	0	0	0
Minimum	0	0	0
Maximum	141	90	7.67
Std. Deviation	7.49	4.12	0.42

7.7.8 Support Tracker

The descriptive statistics for Support Tracker in PPS are shown in Table 34. The statistics for Support Tracker are missing for 228 (11.96% of 1907) projects. The usage of Support Tracker seems to be very low since the medians and modes are zero for both "all issues" and "open issues". And the means are also quite low for both "all issues" and "open issues". The maxima and the standard deviations are not high, either. Thus, there are no projects which utilize the Support Tracker a lot. This usage pattern is confirmed by the monthly statistics. The mean of monthly issues (all issues) is only 0.01, with median and mode being zero. Moreover, the maximum and standard deviation are also quite low.

Table 34: Descriptive Statistics for Support Tracker in PPS

(n = 1679; 228 Missing)	All Issues	Open Issues	Monthly Issues
Mean	0.19	0.09	0.01
Median	0	0	0
Mode	0	0	0
Minimum	0	0	0
Maximum	14	14	0.50
Std. Deviation	1.06	0.63	0.05

7.8 Summary

Section 7 analyzed summary statistics of 48,331 OSS projects and more detailed attributes of 1,907 projects which use Python as one of their programming languages. It answered the empirical questions posed in Section 6 by depicting the portraits of OSS projects in general, and Python projects in specific.

Python projects (PPS) are a good representative of all OSS projects at SourceForge.net (SFS). The PPS is large enough in quantity to ensure sufficient sample size. It covers the full spectrums available in the SFS, in terms of topic, operating system, environment, license, and development status. It encompasses the major parts of the spectrums available in the SFS, in terms of intended audience, programming language, and natural language.

Python projects seem to be omnipresent in all kinds of application areas. The emergence of Python projects at SourceForge.net has been at a high rate (over one new project per day on average) from the inception. But the ages of projects have no significant relationship with either the development status or the development team size of projects.

The empirical study revealed some pitfalls and weaknesses of OSS projects, which are listed below together with suggestions for improvements.

- Small development teams and competing projects. Usually not beneficial to the OSS movement, these phenomena should be studied further by OSS researchers. OSS practitioners should reduce the emergence of these phenomena by promoting collaboration within one project instead of competition among similar projects.

- Low tool usage. Only forums and the CVS are widely used by Python projects. Though some projects utilize the forums, Bug Track, and Patch Track a lot, the volumes of tool usage are generally quite low. The reason for the low tool usage should be studied further by OSS researchers. But the small development team could be one major reason.

- Low percentage of "OS Independent" products. The cross-platform feature of Python should be utilized more extensively among Python projects to create OS independent products.

- Low percentage of products under web environment. Smaller percentage of projects in the PPS are deployed in the web environment than that of projects in the SFS. The rich library of web functionality of Python should be utilized more extensively among Python projects to create more products for the web environment.

8 Conclusion

As an exploratory research that inspires original ideas into a burgeoning research domain, this thesis contributes to both comprehension and practice of OSS. The following sections indicate the limitations of this thesis and discuss the implications of this thesis to both OSS researchers and OSS practitioners. This thesis concludes by summarizing the many fertile research areas in OSS which can benefit from the public archives of OSS projects.

8.1 Limitations

The limitations of this thesis are mainly due to its exploratory characteristics. On the theoretical side, this thesis makes many initiative attempts to introduce new ideas and perspectives into OSS research. Understandably, the resulted theoretical analyses and models are not yet exquisite. They need to be refined in future research.

The assessment framework for OSS projects is an obvious one. Theoretically, the assessment framework for OSS projects proposed in this thesis can be extended to a decision support model for various OSS adopters. Empirically, this framework can be deployed in Focus Case Studies or Comparative Case Studies to evaluate some prominent OSS projects, from the perspective of various kinds of adopters. The killer application in Python – Zope (which is not hosted at SourceForge.net) – can be selected as the first test bed of this assessment framework. It is then beneficial to evaluate the Zope project using this assessment framework from the perspectives of Consumer, Prosumer, Profitor, and Partner.

On the empirical side, the data analyses conducted in this thesis are primarily descriptive statistics with some correlational analysis. As indicated in Section 2.1, this level of analysis is the norm within previous research on OSS in the reviewed literature. The lack of advanced statistical analysis is due to the restriction of source data provided by SourceForge.net. For example, the inconsistency among data prohibits the analysis on some important attributes (e.g., release, CVS) of OSS projects. Moreover, the unavailability of historical data prevents conducting valuable time-series analyses. As mentioned in Section 7, further study into individual projects is required to fully understand the various phenomena in OSS projects.

8.2 Contributions

Although constrained by the above limitations, this thesis has its merits. The strategic level study of OSS adoption and the assessment framework for OSS projects are two major theoretical contributions of this thesis. They have theoretical and practical implications to both OSS researchers and OSS adopters.

Innovative is one of the most significant characteristic of the strategic level study of OSS adoption. OSS researchers can use it to analyze OSS adoption from a higher level than the tactic level prevailed in previous adoption studies. OSS adopters can use it to guide their decision of OSS adoption. This thesis highlights that the contents of OSS adoption should be much richer than the adoption of software products as studied in previous research. Actually, the four components of an OSS project should all be integral parts of OSS adoption. The two novel perspectives of Orientation and Participation and the resulted four types of Strategic Usage of OSS are inspiring for both OSS researchers and OSS adopters. The new classification mechanism of OSS adopters according to their

Strategic Usage of OSS sheds fresh lights on the domain governed by the classical classification mechanism based on the timeliness of adoption. In general, the gist of the strategic level study of OSS adoption can also be beneficial to research on IT adoption and diffusion of innovation. It will open vast domains of new research.

Some significant characteristics of the assessment framework for OSS projects are comprehensive, archive-based, and adopter-aware. This framework is comprehensive since it is project-oriented rather than product-oriented. OSS products are assessed within their context: OSS projects and OSS communities. It contains components that are idiosyncratic to OSS: License Analysis, Source Code Analysis, Evolution Analysis, Community Analysis, and Tool Usage Analysis. It is archive-based because it relies on data obtained from the public archives of OSS projects. It is adopter-aware because the implications of different types of adopters are taken into consideration. It can be employed by both OSS researchers and OSS adopters.

Other theoretical contributions of this thesis include:

- Discovery and demonstration of many fertile research areas in OSS that can benefit from the public archives of OSS projects.
- An initial definition of an OSS project by delineating its four critical components.
- A refined specification of roles played by members of OSS communities.
- Initiative effort to analyze the impacts of programming languages on software projects, and suggestion of fruitful research directions on programming languages based on public archives of OSS projects.

- Pioneering analysis of the phenomena of small development teams, peer teams, and competing projects.
- Formulation of a rich set of propositions for future studies.
- Diagnostic discussion on OSS development model and Software Crisis.
- Constructive discussion on documentary structure of source code.
- Suggestion of the Profitor model as a viable business model based on OSS.
- Discerning critics of selected major studies.
- Categorization of public archives of OSS projects, with indication of potential usage of these archives by exampling pertinent previous studies.
- Classification mechanism of previous quantitative studies on OSS.

The major practical contribution of this thesis is the empirical data analysis of OSS projects based on public archives. It depicts the portraits of OSS projects in general, and Python projects in specific. Python projects as a group have not been studied in previous research. This thesis is pioneering in this sense. Caveats, for pitfalls and weaknesses of OSS projects discovered from the analysis, such as small development teams and competing projects, are announced with suggestions for improvements.

Other practical contributions of this thesis include:

- Emphasis of public archives of OSS projects as a valuable resource to OSS researchers and OSS adopters.
- Demonstration of usage of the public archives of OSS projects as major data source for data analysis.

- Highlight of the usage and indication of weaknesses of the diverse services provided by a major OSS repository to the OSS community.

8.3 Fertile Research Areas

Accomplishing its primary research objectives, this thesis discovered and demonstrated many fertile research areas in OSS that can benefit from the public archives of OSS projects. Most significantly, the strategic level study of OSS adoption and the assessment framework for OSS projects are two prominent research areas that can be further refined. And more empirical analyses can be conducted based on the public archives as demonstrated in Section 7.

The various propositions scattered in the body of this thesis can be used as bases for constructing hypotheses to be tested in future studies. The most significant propositions are put together below for convenience.

Proposition 5: When carrying out License Analysis of OSS projects, Consumers and Profitors will less likely have dominant preference for the types of licenses. Prosumers will more likely prefer GPL-like license, and Partners will more likely prefer BSD-like or dual /multiple licenses.

Proposition 7: Compared with their proprietary competitors, OSS products will generally have lower Total Cost of Ownership.

Proposition 10: When carrying out Execution Analysis of OSS projects, Prosumers and Partners will more likely assign favorable scores to revealed weaknesses than will Consumers and Profitors.

Proposition 15: When carrying out Evolution Analysis of OSS projects, Prosumers will more likely prefer the projects being at an early stage than will Consumers and Profitors.

Proposition 18: When carrying out Community Analysis of OSS projects, Prosumers and Partners will more likely assign favorable scores to revealed weaknesses than will Consumers and Profitors.

Proposition 22: For OSS projects, Vitality of Community is positively related to Sufficiency of Documentation.

Proposition 25: For OSS projects, Vitality of Community is positively related to Sufficiency of Tool Usage.

Proposition 26: There will be fewer developers who are willing to contribute to projects under public domain than developers who are willing to contribute to projects under OSI approved licenses.

Proposition 27: Single Team will be most appropriate for projects with low complexity.

Proposition 28: Peer Team will be most appropriate for projects with medium complexity.

Proposition 29: Hierarchical Team will be most appropriate for projects with high complexity.

Proposition 30: Compared with Peer Team projects and Hierarchical Team projects, Single Team projects will generally have lower Vitality of Community.

Proposition 31: Competing OSS projects will occur more frequently among small development team projects than among large development team projects.

Proposition 32: In order to produce the best OSS products and provide the best OSS services, collaboration within one project will be more cost-efficient than will competition among similar projects aimed for the same or comparable functionality.

Proposition 33: Competing OSS projects will more likely use different programming language sets than use identical programming language sets.

Other useful research areas include the following, in the order of appearance in the thesis:

- Section 2.2 indicates that a more complex and comprehensive classification scheme for OSS research may need to be built up when requirements mature.
- As pointed in Section 2.2.3, the evolution of OSS projects can be studied by carrying out time-series analysis over a large sample of OSS projects.
- The relationship between changes in software modules due to bug fixes and the quality of the modules discussed in Section 3.3 could be insightful in a study to assess the design of modules and the overall architecture of software application.
- Section 3.4, 3.5, and 3.6 briefly compare the popular communication tools like mailing lists, newsgroups, and forums. Obviously, this thesis is inclined to forums. Though Section 7.7.2 provides evidence that forums are actually preferred in Python projects to mailing lists, more solid empirical evidences are needed to reveal which tool will dominate.
- Issues discussed in Section 4.2 about the OSS development speed, cost, and quality can be studied in future research.

- Section 4.3.1 recommends several domains for study on the impacts of programming languages, which is also a topic originated by this thesis.
- Section 4.3.2 suggests several venues for study on Documentary Structure.
- As discussed in Section 6.3.1, research on software process assessment for OSS development model is a void to fill.
- Many research areas are recommended in Section 7. Generally, inspecting individual OSS projects can bring profound findings that are not achievable in a Sample Project Study or a Population Project Study.
- As persuaded in Section 7.3, putting software in the public domain is not a license strategy recommended for OSS. It would be interesting to investigate the minority of OSS projects who adopted this strategy in order to find out the reasons for this strategy and the real impact of this strategy on these projects. Moreover, inspection of OSS projects who adopted dual / multiple license strategy may reveal if this strategy is a smart choice.
- As pinpointed in Section 7.6.1, the "many administrators, no developer" projects are of interest to OSS researchers because of their peer-to-peer trait. Although three kinds of organization for development teams are suggested in this thesis, it is still unknown which kind of organization will fit best in what situation. This is a topic with great potential of benefits for OSS practice.
- As argued in Section 7.6.2, there are several interesting topics on small development teams, especially those one-person teams, and on competing projects. The argument

that collaboration is better than competition has important implications to OSS practice.

As emphasized in this thesis, the public archives of OSS projects are a valuable resource to be further explored. They provide ample opportunities for OSS researchers. Although this thesis suggests many fertile research areas, there are even more jewels in the treasure trove for OSS researchers to discover.

REFERENCES

Anderson, B. B., Bajaj, A., & Gorr, W. (2002). An estimation of the decision models of senior IS managers when evaluating the external quality of organizational software. *The Journal of Systems and Software, 61*, 59-75.

Asklund, U., & Bendix, L. (2002). A study of configuration management in Open Source Software projects. *The IEE Proceedings - Software, 149*(1), 40-46.

Antoniol, G., Villano, U., Merlon, E., & Di Penta, M. (2002). Analyzing cloning evolution in the Linux kernel. *Information and Software Technology, 44*, 755-765.

Berlecon Research (2002). FLOSS final report. *Free / Libre Open Source Software: Survey and study*. Retrieved September 16, 2002, from http: //www.infonomics.nl /FLOSS /report/

Boehm, B. (1981). *Software Engineering economics*. Englewood Cliffs, NJ, USA: Prentice-Hall.

Brooks, F. P. (1995). *The mythical man-month: Essays on Software Engineering.* Reading, MA, USA: Addison Wesley Professional.

Brown, M. C. (2001). *Python: The complete reference.* Berkeley, CA, USA: Osborne / McGraw - Hill.

Capiluppi, A., Patricia, L., & Morisio, M. (2002). Characterizing the OSS process. *24th International Conference on Software Engineering*, (Orlando, Florida, USA).

Chandra, S. & Chen, P. M. (2000). Whither generic recovery from application faults? A fault study using open source software. *Proceedings of the 2000 International*

Conference on Dependable Systems and Networks / Symposium on Fault-Tolerant Computing.

Chun, W. J. (2001). *Core Python programming.* Upper Saddle River, NJ, USA: Prentice Hall.

Crowston, K., & Scozzi, B. (2002). Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development. *The IEE Proceedings - Software, 149*(1), 3-17.

Davis, F. D. (1989). Perceived usefulness, perceived ease-of-use and user acceptance of Information Technology. *MIS Quartly, 13*(3), 319-339.

Davis, M., O'Donovan, W., Fritz, J., & Childress, C. (2000). Linux and open source in the academic enterprise. *Proceedings of Special Interest Group on University and College Computing Services*, 65-69.

Dempsey, B. J., Weiss, D., Jones, P., & Greenberg, J. (2002). Who is an Open Source Software developer? Profiling a community of Linux developers. *Communications of the ACM, 45*(2), 67-72.

DiBona, C., Ockman, S., and Stone, M. (Eds.) (1999). *Open Sources: Voices for the Open Source Revolution.* Sebastapol, CA, USA: O'Reilly.

Dinkelacker, J. & Garg, P. K. (2001). Corporate source: Applying open source concepts to a corporate environment. *23th International Conference on Software Engineering*, (Toronto, Canada).

Emam, K. E., & Jung, H. (2001). An empirical evaluation of the ISO / IEC 15504 assessment model. *The Journal of Systems and Software, 59*, 23-41.

Feller, J. & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. *Proceedings of the 21st International Conference in Information Systems*, 58-69.

Feller, J. & Fitzgerald, B. (2001). Open Source Software: Investigating the Software Engineering, psychosocial and economic issues. *Information Systems Journal, 11*(4).

Ferrin, B. G. & Plank, R. E. (2002). Total Cost of Ownership models: An exploratory study. *The Journal of Supply Chain Management, summer*, 18-29.

Fogel, K. & Bar, M. (2002). *Open Source Development with CVS* (2nd ed.). New York: Coriolis.

Gallivan, M. (2001). Organizational adoption and assimilation of complex technological innovations: Development and application of a new framework. *The Data Base for Advances in Information Systems, 32*(3), 51-85.

Gefen, D. (2000). It is not enough to be responsive: The role of cooperative intentions in MRP II adoption. *The Data Base for Advances in Information Systems, 31*(2), 65-79.

Glass, R. L., Vessey, I., & Ramesh, V. (2002). Research in Software Engineering: An analysis of the literature. *Information and Software Technology, 44*, 491-506.

Godfrey, M. & Tu, Q. (2000). Evolution in open source software: A case study. *Proceedings of IEEE International Conference on Software Maintenance*, 131-142.

Godfrey, M. & Tu, Q. (2001). Growth, evolution, and structural change in open source software. *International Workshop on Principles of Software Evolution*, (Vienna, Austria).

Gonzalez-Barahona, J. M., Perez, M. A. O., Quiros, P. H., Gonzalez, J. C., & Olivera, V. M. (2002). *Counting potatoes: The size of Debian 2.2.* Retrieved September 14, 2002, from http: //people.debian.org /~jgb /debian-counting /counting-potatoes/

Halloran, T. J. & Scherlis, W. L. (2002). High quality and open source software practices. *24th International Conference on Software Engineering*, (Orlando, Florida, USA).

Hankin, C., Nielson, H. R., & Palsberg, J. (1996). Strategic directions in research on programming languages. *ACM Computing Survey*, *28*(4), 644-652.

Harman, M., Munro, M., Hu, L., & Zhang, X. (2002). Source code analysis and manipulation. *Information and Software Technology*, *44*, 717-720.

Hecker, F. (2002). *Setting up shop: The business of Open-Source Software.* Retrieved June 27, 2002, from http: //www.hecker.org /writings /setting-up-shop.html

Hertel, G., Niedner S., & Hermann, S. (2002). Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel. (working paper).

Holden, S. (2002). *Python Web Programming.* Indianapolis, Indiana, USA: New Riders.

Jackson, M. C., Chow, S., & Leitch, R. A. (1997). Towards an understanding of the behavioral intention to use an information system. *Decision Sciences*, *28*(2), 357-389.

Jorgensen, N. (2001). Putting it all in the trunk: Incremental software development in the FreeBSD open source project. *Information Systems Journal*, *11*(4), 321-336.

Karahanna, E., Straub, D. W., & Chervany, N. L. (1999). Information Technology adoption across time: A cross-sectional comparison of pre-adoption and post-adoption beliefs. *MIS Quartly*, *23*(2), 183-213.

Keil, M. Mann, J., & Rai, A. (2000). Why software project escalate: An empirical analysis and test of four theoretical models. *MIS Quartly*, *24*(4), 631-664.

Koch, S. & Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, *12*(1), 27-42.

Krishnamurthy, S. (2002). *Cave or community? An empirical examination of 100 mature Open Source Projects*. Retrieved August 2, 2002, from http: // opensource.mit.edu /papers /krishnamurthy.pdf

Lakhani, K. & von Hippel, E. (2002). *How open source software works: "Free" user-to-user assistance*. MIT Sloan School of Management working paper. MIT, Cambridge, MA, USA. Retrieved August 2, 2002, from http: // opensource.mit.edu /papers /lakhanivonhippelusersupport.pdf

Leung, K. R. P. H., & Leung, H. K. N. (2002). On the efficiency of domain-based COTS product selection method. *Information and Software Technology*, *44*, 703-715.

Lutz, M. (2001). *Programming Python* (2nd ed.). Sebastapol, CA, USA: O'Reilly & Associates.

Madey, G., Freeh, V., & Tynan, R. (2002a). Understanding OSS as a self-organizing process. *24th International Conference on Software Engineering*, (Orlando, Florida, USA).

Madey, G., Freeh, V., & Tynan, R. (2002b). The Open Source Software development phenomenon: An analysis based on social network theory. *8th Americas Conference on Information Systems*, (Dallas, TX, USA).

Mockus, A., Fielding, R. & Herbsleb, J. (2000). A case study of open source software development: The Apache server. *Proceedings of International Conference on Software Engineering*, 263-272.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution patterns of open-source software systems and communities. *International Workshop on Principles of Software Evolution*, (Orlando, Florida, USA).

Ousterhout, J. (1999). Free software needs profit. *Communications of the ACM, 42*(4), 44-45.

Payne, C. (2002). On the security of open source software. *Information Systems Journal, 12*(1), 61-78.

Pepper, P., Cebulla, M., Didrich, K., & Grieskamp, W. (2002). From program languages to software languages. *The Journal of Systems and Software, 60*, 91-101.

Pfaff, B. (1998). Society and open source: Why open source software is better for society than proprietary closed source software. Retrieved July 2, 2002, from http: //www.msu.edu /user /pfaffben /anp /oss-is-better.html

PMI (Project Management Institute) (2000). *A Guide to the Project Management Body of Knowledge (PMBOK)*. Upper Darby, PA, USA: Project Management Institute.

Prechelt, L. (2000). An empirical comparison of seven programming languages. *IEEE Computer, 33*(10), 23-29.

Raymond, E. S. (2000). Why Python? *Linux Journal, 73.*

Raymond, E. S. (2001). The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary (Revised ed.). Sebastapol, CA, USA: O'Reilly & Associates.

Robbins, J. (2002). Adopting OSS methods by adopting OSS tools. *International Conference on Software Engineering*, (Orlando, Florida, USA).

Rogers, E. M. (1995). *Diffusion of Innovations* (4th ed.). New York: The Free Press.

Samdred, J. (2001). *Managing open source projects: A Wiley tech brief.* New York: John Wiley & Sons.

Schach, S. R, Jin, B., Wright, D. R., Heller, G. Z., & Offutt, A. J. (2002). Maintainability of the Linux Kernel. *The IEE Proceedings - Software, 149*(1), 18-23.

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open source software development. *Information Systems Journal, 12*(1), 43-60.

Szajna, B. (1994). Software evaluation and choice: Predictive validation of the technology acceptance instrument. *MIS Quartly, 18*(3), 319-324.

Thong, J. Y. L. (1999). An integrated model of Information Systems adoption in small business. *Journal of Management Information Systems, 15*(4), 187-214.

Van De Vanter, M. L. (2002). The documentary structure of source code. *Information and Software Technology, 44*, 767-782.

van der Hoek, A. (2000). Configuration management and open source projects. *Proceedings of the 3rd International Workshop on Software Engineering over the Internet*, (Limerick, Ireland).

Wang, H. & Wang, C. (2001). Open source software adoption: A status report. *IEEE Software*, March/April, 90-95.

Wheeler, D. A. (2002a). *More than a gigabuck: Estimating GNU/Linux's size*. Retrieved September 14, 2002, from http: //www.dwheeler.com /sloc

Wheeler, D. A. (2002b). *Why Open Source Software / Free software (OSS / FS)? Look at the numbers*. Retrieved September 14, 2002, from http: //www.dwheeler.com /oss_fs_why.html

Zhang, W. & Storck, J. (2001). Peripheral members in online communities. *Americas Conference on Information Systems*, (Boston, MA, USA)

Zhao, L. & Elbaum, S. (2000). A survey on quality related activities in open source. *Software Engineering Notes*, *25*(3), 54-57.

APPENDICES

Appendix A. The Open Source Definition

Version 1.9 (http: //www.opensource.org /dos /definition_plain.php)

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and

used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. The License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

Appendix B. Overview of SourceForge.net Services

(https: //sourceforge.net /docman /display_doc.php?docid=753&group_id=1)

This document contains an overview of the services provided to projects hosted at SourceForge.net. SourceForge.net provides free hosting services for Open Source projects. This services list continues to be expanded as we add new services and improve on existing ones.

Additional details regarding the services listed on this page may be found in the SourceForge.net site documentation collection. Should you have questions or concerns regarding this document, please submit a support request.

As follows is a list of the services currently provided by SourceForge.net:

SourceForge Collaborative Development System (CDS) web tools

Advanced web-based tools provide easy maintenance of all facets of your project. These tools are located on the Project Admin page for your project, accessible from the summary page for your project. Through the project administration interface provided to SourceForge.net projects, you can post news items, easily administrate your development team, post "help wanted" items related to your project, and manage the use of services provided through the SourceForge.net web site. The software used to run the SourceForge.net site is an older version of the SourceForge software platform, as provided by VA Software, Inc.

Project Web Server

http://projectname.sourceforge.net

Clearly one of the most important parts to managing your project is providing potential users and existing users with information about your project. To aid you in reaching this goal, we provide you with your own virtual host on our project web servers. Each project has their own space for web content and CGI scripts. PHP scripts are also supported on the project web server, allowing you to build a more refined web presence for your project. Each project is provided with up to 100MB of space to host web content on their project web server.

Tracker: Tools for Managing Support

SourceForge.net provides a suite of SourceForge-integrated support management tools, called Tracker. Tracker provides bug reporting facilities, the means for your users to submit support requests, the means for developers to easily submit patches for your review, and a suggestion box where people can post feature requests.

These tools are extremely easy to use, both as a developer and an end-user; requiring a minimal amount of time both to post issues and to respond to them. Users and developers alike are updated automatically via e-mail when one of their tracker items has been updated, keeping everyone well informed as to the status of their issue or request.

An excellent number of options are provided for classification of issues and requests. Project administrators may review statistical details of project support, including details related to response times for issue handling, and the breakdown of requests by classification. These tools truly provide an excellent means to provide an exceptional level of support to your users and manage ongoing product development.

Mailing lists and discussion forums

How can you stay in touch with your development team and your user base? SourceForge.net provides projects with mailing list services and web-based discussion forums. The ability to administrate your mailing list is provided via a web interface. Public mailing lists hosted by SourceForge.net are archived within our integrated mailing list archival system, providing you and your users with easy access to previous list traffic from a web interface. We have also taken the time to implement a strong spam prevention system, reducing the amount of offtopic and unwanted mailing list traffic.

Releasing your software

Projects hosted at SourceForge.net may release their software through the use of a web-based file release system. Releasing your software through SourceForge.net provides you with greater visibility, and the means to track the number of times your software is downloaded. Software released through SourceForge.net's file release system is placed on a network of high-capacity, high-performance download servers, capable of handling well in excess of 1000 concurrent FTP connections and 3000 concurrent HTTP connections.

Users may monitor your packages to receive automatic e-mail notification when a new release becomes available. You may also choose to include release notes and changelog information with your releases, ensuring that your user base has easy access to the information they need.

Shell services and compile farm

Developers on active SourceForge.net projects are provided with access to a shell account via SSH. This shell may be used for basic shell functions, has its own crontab, and may be used to directly manipulate the web content for your project.

SourceForge.net also provides access to a diverse network of hosts running a variety of operating systems. The SourceForge.net compile farm may be used to test your software on operating systems you may not have direct access to otherwise, and to generate pre-built binaries for these platforms if you so choose.

MySQL Database Services

Each project is, upon their request, provided with their own MySQL database. Projects may use this database for development and testing, or to drive a component of their project web site. Project MySQL databases may be accessed from PHP scripts on the project web server, allowing you to provide enhanced features and (for certain types of projects) demos of your releases.

Project CVS Services

Each project hosted at SourceForge.net is provided with their own CVS repository to aid in further development of your project. Developers on your project are automatically granted write access to your project CVS repository via SSH. Anonymous, read-only pserver-based access to your repository is provided to the general public. You may also view and compare the contents of your CVS repository through the use of a web-based interface, provided from your project CVS page (accessible from the summary page for your project).

VHOST Services

You may request (via a web-based interface on the Project Admin page for your project) that we answer web traffic for your registered domain. (DNS services are not currently provided to SourceForge.net projects.)

Trove Listing

Would you like to increase the visibility of your project? Projects hosted at SourceForge.net may classify themselves in the Trove, a massive database of Open Source projects. The Trove is searchable from the SourceForge.net site. By categorizing your project within the Trove, you help users interested in software like yours to find it, quickly and easily.